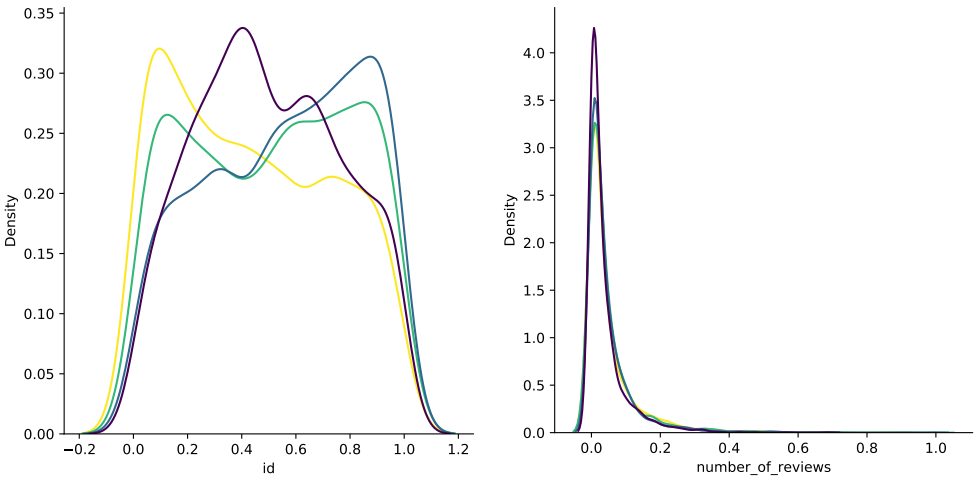
KAGGLE COMPETITION WRITE-UP

Braden Hoagland
Kaggle username: bradenhoagland

COMPSCI 671: Machine Learning

Exploratory Analysis

To begin exploring the training data, I created density plots for the values of each feature. Figure 1 shows two of these plots.



(a) id

(b) number of reviews

Fig. 1: Density plots for two of the features. Purple lines correspond to the density of price label 1, blue for price 2, green for price 3, and yellow for price 4.

After making the density plots, it was obvious that the 'id' feature was random. I also noticed that the 'is_business_travel_ready' had only one value. As a result, I removed both of them from the dataset.

Additionally, I used these density plots to experiment with setting thresholds on the data. Note that with the number_of_reviews feature, almost all values are less than 0.4 (note that these quantities were already normalized, thus why the number of reviews is less than 1). One way of removing outliers from the data would be to set a threshold at 0.4. If a data point's number of reviews falls above 0.4, then it is removed from the dataset. I created thresholds like this for all features that seemed to have skewed distributions.

My tests with these "outliers" removed yielded results that were strictly worse than when I included all the datapoints, which insinuates that I was not lenient enough with my thresholds. Because of the large number of features that I would have to manually tune, I focused on other forms of data analysis instead of this method of outlier removal.

I then examined the correlation between the different features. Figure 2 shows a heatmap that represents the correlation matrix, where the x and y-axes are the same: namely, they are each composed of every (non-removed) feature. I did not include the axis ticks because of formatting issues in this document, but I was able determine which of the features had the largest correlation. These pairs are:

- reviews_per_month and number_of_reviews
- require_guest_profile_picture and require_guest_phone_verification
- bedrooms and bathrooms and beds
- guests_included and beds, and
- host_is_superhost and reviews_per_month.

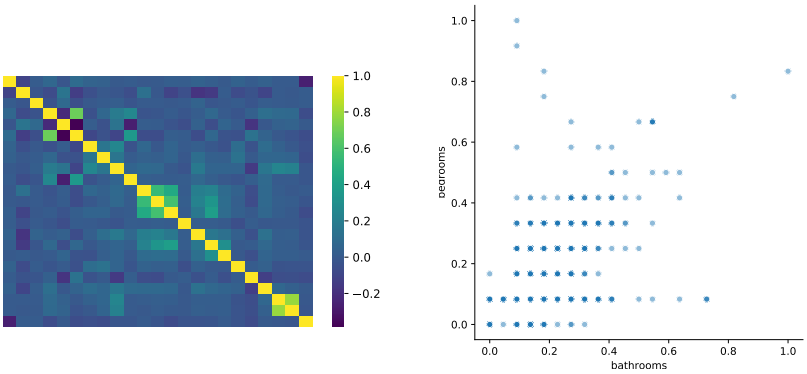


Fig. 2: The coorrelation for all remaining features, as well as an example of the correlation between bathrooms and bedrooms.

I noted these correlations, and later on during training ran ablation tests with different pairs of correlated variables removed. I then kept the features that yielded the best results. I ultimately decided to remove number_of_reviews and require_guest_phone_verification from the dataset, as these yielded the models with the best validation accuracies.

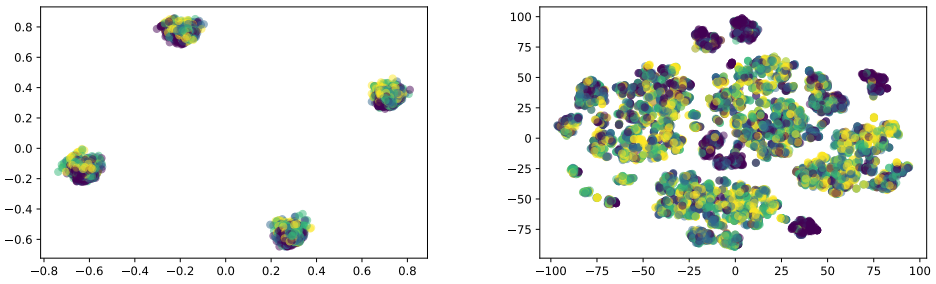
Feature Engineering

I then tried to engineer some new features for the dataset that could take into account extra domain knowledge. The first feature I tried to construct was a notion of population density in each neighborhood. I determined the longitude and latitude of each neighborhood, and planned on comparing these to population density data from Buenos Aires in order to create a new 'population_density' feature for each data point.

However, I was unable to actually find population density data for Buenos Aires, so instead I calculated the length of the geodesic between each neighborhood and the center of the city. My intuition here was that the population density should be correlated with the distance to the center of the city. This gave me a new 'distance' feature that I appended to all data points and used for the remainder of my experiments.

Data Clusters

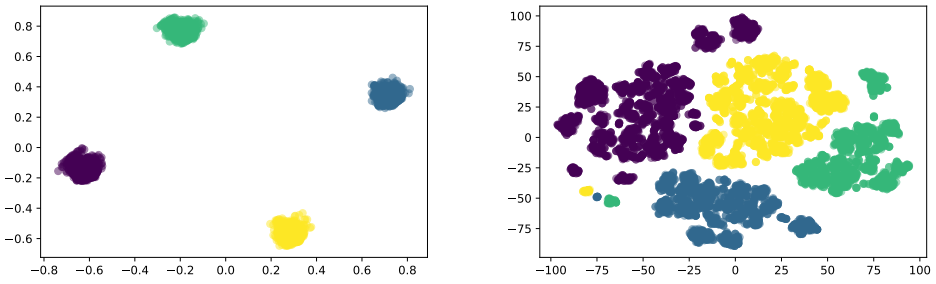
I wanted to experiment with clustering the data as a means of preprocessing, so I visually examined the data using the t-SNE and PCA algorithms. Figure 3 shows the result of running both algorithms on the data to embed it into 2 dimensions.



(a) PCA (b) t-SNE

Fig. 3: Two-dimensional embeddings of the training data using the PCA and t-SNE algorithms, where color corresponds to the true price label.

Both seem to show 4 potential clusters, so I ran the K-means algorithm (implementation details are in the Training section of this report) on the data with $K = 4$ to divide the data into 4 parts. Figure 4 shoes the data embeddings again, but each point is colored according to its cluster assignment instead of its true price label.



(a) PCA (b) t-SNE

Fig. 4: Two-dimensional embeddings of the training data using the PCA and t-SNE algorithms, where color corresponds to the assigned cluster.

This seems like a good qualitative match with the data, so I saved the cluster centers. I planned on creating 4 separate classifiers - 1 for each cluster. To make a price prediction for a single data point, I would determine its nearest cluster center, then use that cluster's classifier in order to make a prediction. My intuition here was that I could perhaps remove some of the complexity in the relationship between data and labels by manually splitting up data that was very different. This might allow each of the 4 individual models to pick up on local patterns in the data that were **not** global patterns in the data.

Neighborhood Clusters

In addition to clustering the data as a whole, I also examined the geographic locations of the neighborhoods. Figure 5 shows the coordinates of each neighborhood, where the x-axis is longitude and the y-axis is latitude. Unlike with the data as a whole, there was no clear number of clusters for this data. To protect against overfitting, I set the number of clusters to 3, which seems to split the data without making any of the clusters too small. Similar to the training method used with the previous clustering strategy, I

planned on dividing up the data based on which neighborhood cluster it belonged to, then using that to determine which of three classifiers it should use for its prediction. My intuition here was that geographically proximate neighborhoods should have similar ppopulation densities, and thus should have similar styles of housing, similar patrons, and, similar prices distributions.

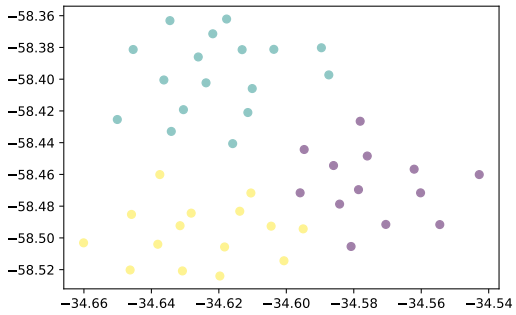


Fig. 5: The geographic locations of the neighborhoods from the training data, split into three clusters.

Models

With such complex data, I assumed that the relationship between the data and the labels would be non-linear. Thus, methods like SVM with RBF kernel and decision trees seemed like appealing choices. I was wary of using SVM, as I did not want to overfit to the training data, but still wanted to take advantage of their expressive power.

The two models I settled on were

(1) an SVM model with RBF kernel, with parameters $\gamma = 0.5$ and $C = 50$; and
(2) a random forest, where each individual decision tree had a maximum depth of 20.

Although I did test standard decision trees, I used random forests for my main model instead because of their greater expressive power. Details of how I chose the aforementioned hyperparameters are in the Hyperparameter Selection section of this report.

I decided on using standard random forests instead of a cluster-based variant for my final model after significant testing. Since the cluster-based hybrid models were a significant amount of my work on this project, though, I still include details of how I trained them and selected hyperparameters later on.

Another factor in my decision to use these two families of models is the existence of easy-to-use libraries implementing both models. I used the Scikit-Learn library to create and train my models. For the SVM, I was able to use the library's 'svm.SVC' model directly and only worry about changing hyperparameters. For the cluster-random forest hybrid models, I was able to create a custom Scikit-Learn classifier that managed multiple random forests at once and used pre-determined clusters to decide which classifier to use when making a prediction on a given data point. The random forests in my cluster models were implemented using the 'ensemble.RandomForestClassifier' model from Scikit-Learn.[2]

Since both of my families of models were implemented in this way, I was able to use standard Scikit-Learn cross validation functions on both, which I go into more detail on in the Data Splits section of this report.

Training

Training Algorithms

The Scikit-Learn SVM model that I used implements multi-class classification by training 6 classifiers (the number of combinations of 2 classes when there are 4 classes total). It is able to decide what class a given point belongs to by running the two-class classifiers on each combination of classes and choosing the single class that beats out the rest.

In order to train the SVM in the first place, the library approximately solves the primal optimization problem

$$\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i$$

such that $y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i$ and $\zeta_i \geq 0$ for all i. In this case, ϕ is a map from our given feature space to the feature space under the RBF kernel and ζ_i is the error for point i.

In order to solve this, the library instead solves the dual optimization problem

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha,$$

such that $y^T \alpha = 0$ and $0 \leq \alpha_i \leq C$ for all i. In this formulation, $Q_{ij} = y_i y_j K(x_i, x_j)$, where K is the RBF kernel.

In order to actually solve this dual problem, Scikit-Learn uses the libsvm library, which itself uses Sequential Minimal Optimization (SMO) to solve the dual.[1]

The Scikit-Learn RandomForestClassifier constructs decision trees based on bootstrap samples from the training data. The splits in each decision tree in the forest are allowed to choose from $\sqrt{n}$ features, where n is the total number of features. The splits are chosen to maximize the Gini index.

Each decision tree outputs a probabilistic answer. If a leaf node in a decision tree has proportion p of class 1 and if p is the largest proportion of the classes, then the decision tree(assuming it reaches this particular leaf node) will output class 1 along with p. This gives each prediction a sense of certainty. When combining the predictions from each tree in the forest, the Scikit-Learn implementation combines these probabilities instead of just counting how many trees voted for which class.

Runtime

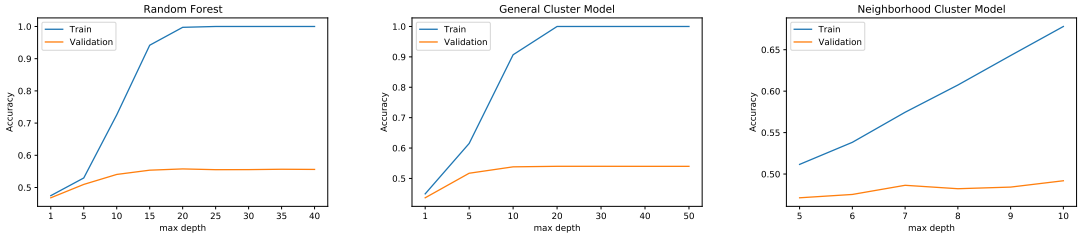
In terms of wall time, the random forest method was the fastest to train, taking approximately one minute to run through parameter sweeps on my laptop. The SVM model was slower, taking anywhere between 1 and 5 minutes, depending on how many other tasks were currently running on my computer. The cluster-based hybrid models took significantly longer, mainly due to an inefficient prediction function. They frequently took longer than 10 minutes to train when using parameter sweeps.

In order to train just a single model with no parameter sweeps, the random forest took a matter of seconds, SVM took a decent fraction of a minute, and the cluster-based model frequently took over a minute.

Hyperparameter Selection

In order to select hyperparamters, I used a series of searches across what seemed like reasonable chunks of parameter space. For my random forest models, I only varied the maximum depth parameter. To find the optimal parameter, I fixed all others and ran K-fold[a] cross validation on models with varied values of the maximum depth parameter. I selected the parameter value with the maximum validation accuracy.

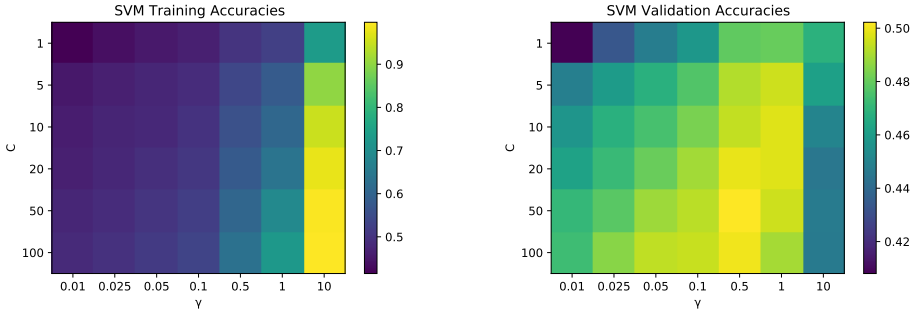
[a]For the cluster-based random forest methods, I only used 3 folds because of computational constraints. For the standard random forest model, I used 10 folds.



(a) (b) (c)

Fig. 6: Training and validation curves for both cluster-based hybrid models: (a) Random Forest, (b) General Cluster-based model, and (c) neighborhood cluster-based model.

For the SVM model, I tuned both the RBF kernel parameter γ and the regularization coefficient C. Instead of just varying one of the constants, I created a grid of possible parameter values, where each square in the grid represented a different combination of parameters. I used cross validation on a model for each square in the grid, and chose the parameters from the square with the highest validation accuracy. Figure 7 shows the training and validation accuracies from this process.



(a) Training (b) Validation

Fig. 7: Training and Validation accuracies from the cross validation scheme for the SVM model.

Data Splits

All of my models used the same cross validation scheme to determine model performance. I divided the training set into K folds of equal size, and cycled through the folds, setting one to be the validation fold and the others to be the training folds. To evaluate a model, I trained it using the training folds and tested it using the validation fold. I repeated this for each fold cycle and averaging the training and testing accuracies to get my final performance metrics. This process was implemented using the Scikit-learn 'KFold' method.

Although I kept track of the training accuracies in this process, the validation/testing accuracy was more important, as this gave me a sense of how well my models generalized to unseen data, i.e. how well they avoided overfitting.

Errors and Mistakes

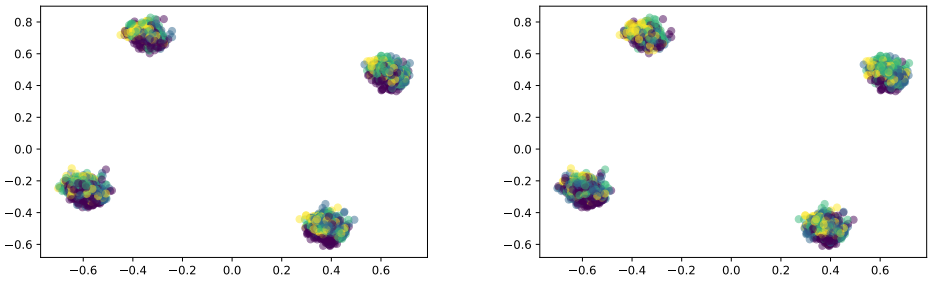
Throughout my experiments, I ran into the common pitfall of not managing the variables in my Jupyter notebooks neatly enough. Every so often, I would realize that I was using a variable name that I had deleted, but my code was not failing because that particular variable still existed in my current session. This is a somewhat artificial error, but I needed to mention it nonetheless since it did delay my progress by a non-trivial amount.

The biggest mistake I made during this project was probably trying to dive right into the model development phase without spending enough time interacting with the data. Originally, I just converted all my data to floating point numbers and then began experimenting with SVM and random forest models. After about a day of this, I realized that I did not understand what my data was, and I went back to the data preparation stage. It was only then that I was able to do feature engineering and determine correlated features.

The hardest part of this project was being creative with unfamiliar data. In earlier assignments, if we needed to use data, it was straightforward and worked out-of-the-box. This time around, however, I had to manipulate and analyze the data in order to make it more informative. This was my first experience with feature engineering and in-depth data analysis, so it required a lot of experimentation on my part.

Predictive Accuracy

The results in this section correspond to the testing accuracy displayed prior to the end of the competition, which was calculated using only a portion of the full test set. My final results may be slightly different than these. For reference, my Kaggle username is bradenhoagland.



(a) Random Forest

(b) SVM

Fig. 8: Model predictions on the test set. The random forest predictions shown come from the non-clustering model.

My SVM model, even with correlated features removed, was only able to achieve 52% test accuracy. Although its training accuracy was close to perfect, the expressive power of the RBF kernel was not limited enough for it to generalize well. I could have potentially improved the generalization ability of this model through more hyperparamter tuning, but the results from my hybrid models were promising enough that I focused on those instead.

Before removing correlated features, my standard random forest model was able to achieve 55% test accuracy. After removing corrleated features, this became 56%. Perhaps not surprisingly, adding clusters resulted in poorer generalization, with my general cluster model having 54.5% test accuracy and my neighborhood cluster model having 49.5% test accuracy. The general clusters gave the random forest model better training accuracy, but the neighborhood clusters resulted in both poorer trianing and poorer validation accuracy. The accuracy curves in Figure 6 from the Hyperparameter Selection section of this report support this claim.

Figure 8 shows 2-dimensional PCA embeddings of the test set, where the points are colored based on the class prediction of the respective model. Both models can be seen to be very similar, with only slight differences in some areas of each cluster.

Code

Below are the two Jupyter notebooks I used for this project. Almost all the outputs should be there, with the one major exception of the density plots that I used during my exploratory data analysis. They took up almost 10 pages on their own, so I removed them from the output.

The first PDF is my main training notebook, which has almost all of the results. The second PDF is the notebook I used to train my neighborhood cluster-based model. The training pipeline for this model was different, thus a second notebook. As a result, much of the code in the second notebook can also be found in the first notebook.

NOTEBOOK 1: MAIN

In [1]:
```
import sys
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# random seeds
np.random.seed(0)
```

Progress Bar

In [2]:
```
def out(action, bar, percent, done=False):
    done = f'\n' if done is True else ''
    sys.stdout.write('\r%s |%s| ' % (action.ljust(30), bar) + '{0:.0f}%'
.format(percent) + done)

def progress(i, total, action):
    i = i + 1
    ratio = i / total
    percent = 100 * ratio
    filled = int(round(20 * ratio))

    bar = '' * filled + '-' * (20 - filled)

    done = (percent == 100)
    out(action, bar, percent, done)

    if i == total:
        sys.stdout.write('\n')

    sys.stdout.flush()
```

Import Data

In [3]:
```
X_train = pd.read_csv('data/train.csv')
y_train = X_train['price']
X_test = pd.read_csv('data/test.csv')

features = X_train.columns
labels = list(np.unique(y_train))

string_features = {
    'neighbourhood': ['Agronomía', 'Almagro', 'Balvanera', 'Barracas',
'Belgrano',
       'Boedo', 'Caballito', 'Chacarita', 'Coghlan', 'Colegiales',
       'Constitución', 'Flores', 'Floresta', 'La Boca', 'La Paternal',
       'Liniers', 'Mataderos', 'Monserrat', 'Monte Castro',
       'Nueva Pompeya', 'Núñez', 'Palermo', 'Parque Avellaneda',
       'Parque Chacabuco', 'Parque Chas', 'Parque Patricios',
       'Puerto Madero', 'Recoleta', 'Retiro', 'Saavedra', 'San Cristóba
l',
       'San Nicolás', 'San Telmo', 'Versalles', 'Villa Crespo',
       'Villa Devoto', 'Villa General Mitre', 'Villa Luro',
       'Villa Ortúzar', 'Villa Pueyrredón', 'Villa Real',
       'Villa Santa Rita', 'Villa Urquiza', 'Villa del Parque',
       'Vélez Sársfield'],
    'room_type': ['Entire home/apt', 'Hotel room', 'Private room', 'Shar
ed room'],
    'host_is_superhost': ['f', 't'],
    'bed_type': ['Airbed', 'Couch', 'Futon', 'Pull-out Sofa', 'Real Bed'
],
    'instant_bookable': ['f', 't'],
    'is_business_travel_ready': ['f'],
    'cancellation_policy': ['flexible', 'moderate', 'strict_14_with_grac
e_period', 'super_strict_30', 'super_strict_60'],
    'require_guest_profile_picture': ['f', 't'],
    'require_guest_phone_verification': ['f', 't'],
}
```

Feature Engineering: Distance from city center

In [4]:
```
from geopy import distance
def dist_from_center(coords):
    buenos_aires_center = (-34.603722, -58.381592)
    return distance.distance(buenos_aires_center, coords).km

coords = {
    'Agronomía': (-34.5950, -58.4943),
    'Almagro': (-34.6114, -58.4210),
    'Balvanera': (-34.6101, -58.4059),
    'Barracas': (-34.6454, -58.3813),
    'Belgrano': (-34.5621, -58.4567),
    'Boedo': (-34.6305, -58.4192),
    'Caballito': (-34.6159, -58.4406),
    'Chacarita': (-34.5860, -58.4544),
    'Coghlan': (-34.5602, -58.4716),
    'Colegiales': (-34.5760, -58.4484),
    'Constitución': (-34.6261, -58.3860),
    'Flores': (-34.6375, -58.4601),
    'Floresta': (-34.6282, -58.4844),
    'La Boca': (-34.6345, -58.3631),
    'La Paternal': (-34.5959, -58.4716),
    'Liniers': (-34.6463, -58.5202),
    'Mataderos': (-34.6601, -58.5031),
    'Monserrat': (-34.6131, -58.3814),
    'Monte Castro': (-34.6183, -58.5057),
    'Nueva Pompeya': (-34.6501, -58.4254),
    'Núñez': (-34.5428, -58.4601),
    'Palermo': (-34.5781, -58.4265),
    'Parque Avellaneda': (-34.6459, -58.4852),
    'Parque Chacabuco': (-34.6341, -58.4329),
    'Parque Chas': (-34.5842, -58.4787),
    'Parque Patricios': (-34.6363, -58.4005),
    'Puerto Madero': (-34.6177, -58.3621),
    'Recoleta': (-34.5874, -58.3973),
    'Retiro': (-34.5896, -58.3802),
    'Saavedra': (-34.5545, -58.4916),
    'San Cristóbal': (-34.6238, -58.4023),
    'San Nicolás': (-34.6037, -58.3812),
    'San Telmo': (-34.6218, -58.3714),
    'Versalles': (-34.6308, -58.5208),
    'Villa Crespo': (-34.5947, -58.4443),
    'Villa Devoto': (-34.6007, -58.5144),
    'Villa General Mitre': (-34.6105, -58.4717),
    'Villa Luro': (-34.6381, -58.5040),
    'Villa Ortúzar': (-34.5786, -58.4696),
    'Villa Pueyrredón': (-34.5808, -58.5054),
    'Villa Real': (-34.6197, -58.5240),
    'Villa Santa Rita': (-34.6138, -58.4832),
    'Villa Urquiza': (-34.5705, -58.4915),
    'Villa del Parque': (-34.6045, -58.4926),
    'Vélez Sársfield': (-34.6315, -58.4923)
}

dists = {
    nhood: dist_from_center(coords[nhood]) for nhood in coords
}
```

In [5]:
```
# set new "distance" column to be all 0s
X_train['distance'] = 0.0
X_test['distance'] = 0.0

# fill in the distances from the city center
for i in range(X_train.shape[0]):
    X_train.at[i, 'distance'] = dists[X_train.at[i, 'neighbourhood']]
for i in range(X_test.shape[0]):
    X_test.at[i, 'distance'] = dists[X_test.at[i, 'neighbourhood']]
```

Results Saving

In [6]:
```
train_ids = X_train['id']
test_ids = X_test['id']
```

In [7]:
```
def save_predictions(clf, filename):
    y_pred = clf.predict(X_test)
    assert(y_pred.shape[0] == 4149)

    y_pred = pd.DataFrame(data=y_pred, columns=['price'])
    y_pred['id'] = test_ids
    y_pred = y_pred.reindex(['id','price'], axis=1).astype(int)

    y_pred.to_csv(f'results/{filename}.csv', index=False)
```

Normalize Data

In [8]:
```
from datetime import date

def data2float(X):
    X = X.copy()

    # turn strings into integers
    for f in string_features:
        for i in range(X.shape[0]):
            X.at[i, f] = string_features[f].index(X.at[i, f])

    # turn dates into days relative to today
    today = date.today()
    for f in ['last_review', 'host_since']:
        for i in range(X.shape[0]):
            m, d, y = (int(x) for x in X.at[i, f].split('/'))
            X.at[i, f] = (today - date(y, m, d)).days

    return X.astype(float)

X_train = data2float(X_train)
X_test = data2float(X_test)
```

In [9]:
```
from sklearn.preprocessing import MinMaxScaler

def normalize_df(df):
    min_max_scaler = MinMaxScaler()
    return pd.DataFrame(min_max_scaler.fit_transform(df.values), columns
=df.columns)

X_train = normalize_df(X_train)
X_test = normalize_df(X_test)
```

Cross Validation code

In [10]:
```
from sklearn.model_selection import KFold
from sklearn.base import clone

def cv(clf, X, y, k):
    X = X.copy()
    y = y.copy()
    base_clf = clone(clf)

    train_scores = []
    val_scores = []

    kf = KFold(n_splits=k)
    kf.get_n_splits(X)

    for train_index, val_index in kf.split(X):
        # split data into training and validation
        X_train, X_val = X.loc[train_index], X.loc[val_index]
        y_train, y_val = y.loc[train_index], y.loc[val_index]

        # reset indices for the data
        X_train = X_train.reset_index(drop=True)
        X_val = X_val.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        y_val = y_val.reset_index(drop=True)

        # train and score a classifier
        clf = clone(base_clf)
        clf.fit(X_train, y_train)
        train_scores.append(clf.score(X_train, y_train))
        val_scores.append(clf.score(X_val, y_val))

    train_mean = sum(train_scores) / len(train_scores)
    val_mean = sum(val_scores) / len(val_scores)
    return train_mean, val_mean
```

In [11]:
```
def vary_cv(clf_fn, prop, values, X, y, k):
    train_scores = []
    val_scores = []

    i = 0
    for v in values:
        progress(i, len(values), f'{prop}: {v}')
        i += 1

        clf = clf_fn(v)
        train_mean, val_mean = cv(clf, X, y, k=k)
        train_scores.append(train_mean)
        val_scores.append(val_mean)

    plt.plot(train_scores, label='Train')
    plt.plot(val_scores, label='Validation')
    plt.xticks(ticks=range(len(train_scores)), labels=values)
    plt.xlabel(prop)
    plt.legend()
    plt.show()

    return train_scores, val_scores
```

Data Inspection

Finding bad features

To start, we'll look at the distributions of the values of each feature. From this we can get a sense of which features are correlated with each price.

In [ ]:
```
for f in features:
    sns.displot(data=X_train, x=f, hue='price', kind='kde', palette='vir
idis', legend=False)
    plt.savefig(f'img/density/{f}.pdf')
    #### purple=1, yellow=4
```

In [13]:
```
X_train = X_train.drop(columns=['price'])

for f in ['id', 'is_business_travel_ready']:
    X_train = X_train.drop(columns=[f])
    X_test = X_test.drop(columns=[f])
```

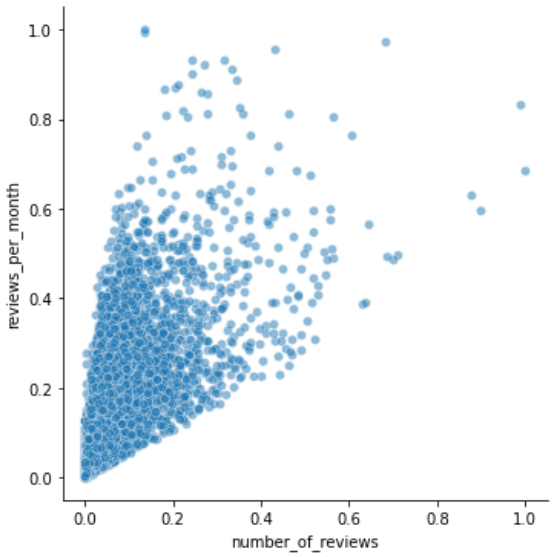
Remove outliers

In [14]:
```
# thresholds = [
#     'minimum_nights < 0.2',
#     'number_of_reviews < 0.4',
#     'last_review < 0.4',
#     'calculated_host_listings_count < 0.4',
#     'bathrooms < 0.4',
#     'bedrooms < 0.4',
#     'beds < 0.4',
#     'bed_type > 0.9',
#     'cleaning_fee < 0.4',
#     'guests_included < 0.4',
#     'extra_people < 0.2',
#     'maximum_nights < 0.1'
# ]

# def remove_outliers(X, y):
#     for t in thresholds:
#         X = X.query(t)
#     y = y.loc[X.index]
#     return X.reset_index(drop=True), y.reset_index(drop=True)

# X_train, y_train = remove_outliers(X_train, y_train)
```

Variable Correlation

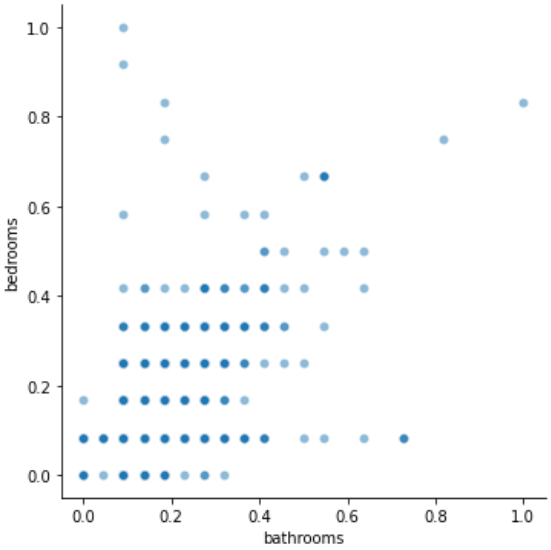
In [15]:
```
sns.relplot(x='number_of_reviews', y='reviews_per_month', data=X_train,
alpha=0.5)
# plt.savefig('img/corr/reviews.pdf')
```

Out[15]: <seaborn.axisgrid.FacetGrid at 0x1401fdcd0>

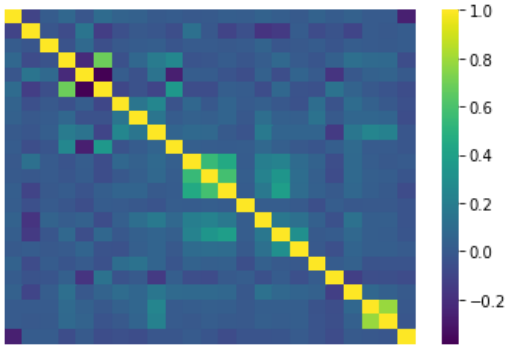


In [16]:
```
sns.relplot(x='bathrooms', y='bedrooms', data=X_train, alpha=0.5)
# plt.savefig('img/corr/reviews.pdf')
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x140280220>



In [17]:
```
corr_matrix = X_train.corr()
sns.heatmap(corr_matrix, cmap='viridis', xticklabels=False, yticklabels=
False)
# plt.show()
# plt.tight_layout()
# plt.gcf().subplots_adjust(bottom=0.3)
plt.savefig('img/corr/matrix.pdf')
```



Relationships

The more &'s, the stronger the correlation:

- reviews_per_month - number_of_reviews - last_review (&&&)
- bedrooms - bathrooms - beds (&&)
- guests_included - beds (&)
- host_is_superhost - reviews_per_month (&)
- require_guest_profile_picture - require_guest_phone_verification (&&&)

Note that 'distance' isn't strongly correlated with anything. That means we've added in new information

K-Means

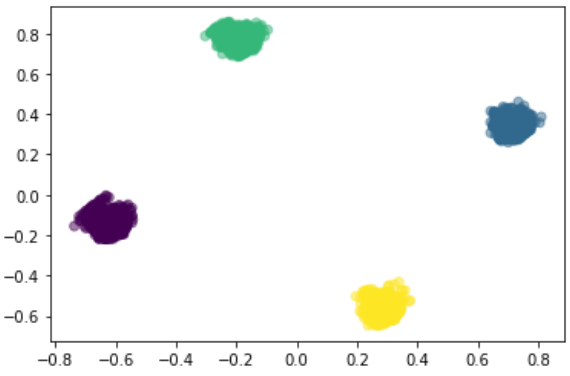
In [18]:
```
from sklearn.cluster import KMeans

n_clusters = 4
kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X_train)
```

In [19]:
```
from sklearn.decomposition import PCA

X2 = PCA(n_components=2).fit_transform(X_train)
```

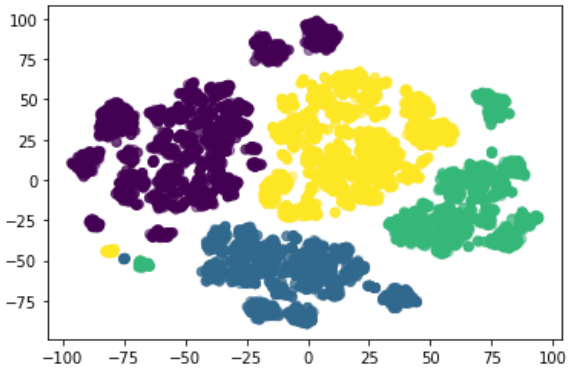
In [20]:
```
from sklearn.manifold import TSNE

X_embed = TSNE(n_components=2).fit_transform(X_train)
```

In [21]:
```
plt.scatter(X2[:,0], X2[:,1], c=kmeans.labels_, alpha=0.5)
# plt.savefig('img/clusters/clusters.pdf')
```

Out[21]:
```
<matplotlib.collections.PathCollection at 0x1411c3400>
```

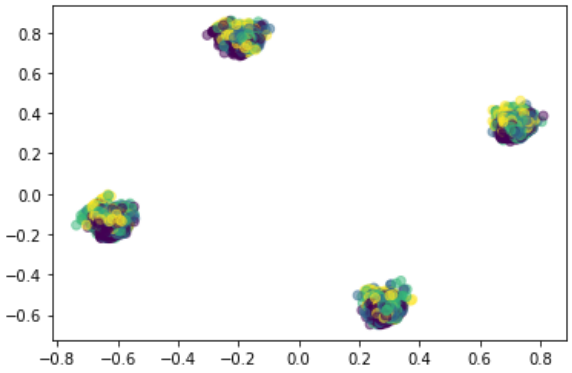


In [25]:
```
plt.scatter(X_embed[:,0], X_embed[:,1], c=kmeans.labels_, alpha=0.5)
# plt.savefig('img/clusters/tsne-clusters.pdf')
```

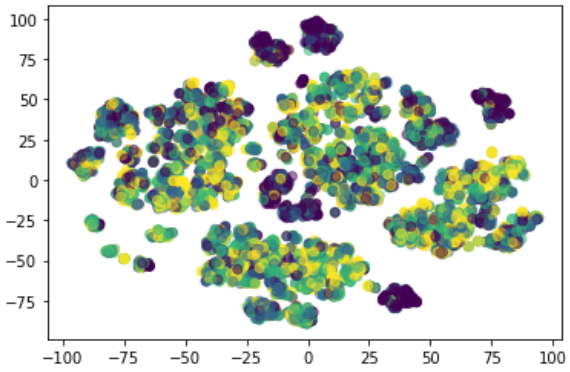


In [23]:
```
plt.scatter(X2[:,0], X2[:,1], c=y_train, alpha=0.5)
# plt.savefig('img/clusters/labels.pdf')
```

Out[23]: <matplotlib.collections.PathCollection at 0x140525820>



In [26]:
```
plt.scatter(X_embed[:,0], X_embed[:,1], c=y_train, alpha=0.5)
# plt.savefig('img/clusters/tsne-labels.pdf')
```



Experimental: K-Means -> Other Model

In [17]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.svm import SVC


class HybridClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, n_clusters, max_depth):
        self.n_clusters = n_clusters
        self.max_depth = max_depth
        self.kmeans = None

    def _get_data_clusters(self, kmeans, X, y):
        cluster_data = {i: () for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            idx = [i for i in range(len(X)) if kmeans.labels_[i] == c]
            cluster_data[c] = (X.loc[idx], y.loc[idx])
        return cluster_data

    def fit(self, X, y):
        # divide data into clusters
        self.kmeans = KMeans(n_clusters=self.n_clusters, random_state=0)
.fit(X)
        cluster_data = self._get_data_clusters(self.kmeans, X, y)

        # fit model for each cluster
        self.cluster_models = {i: None for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            X, y = cluster_data[c]
            clf = RandomForestClassifier(max_depth=self.max_depth, rando
m_state=0)
#             clf = AdaBoostClassifier(n_estimators=self.max_depth, rand
om_state=0)
#             clf = SVC(kernel='rbf', gamma=self.max_depth)
            clf.fit(X, y)
            self.cluster_models[c] = clf

    def predict(self, X):
        if self.cluster_models is None:
            raise AssertionError('You have to train the model first, du
h')

        pred_clusters = self.kmeans.predict(X)

#         pred_y = [self.cluster_models[pred_clusters[i]].predict([X.loc
[i]]) for i in range(len(X))]
        pred_y = []
        for i in range(len(X)):
            x = X.loc[i]
            c = pred_clusters[i]
            pred_y.append(self.cluster_models[c].predict([x]))
        pred_y = np.array(pred_y).flatten()
        return pred_y

    def score(self, X, y):
        pred_y = self.predict(X)
        num_correct = sum([1 if pred_y[i] == y.loc[i] else 0 for i in ra
```

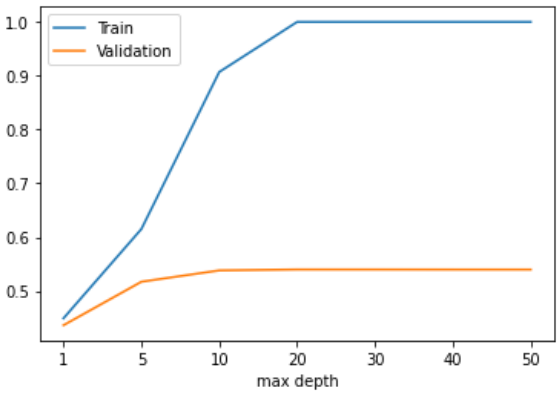
```
nge(len(X))])
        return num_correct / len(X)
```

In [26]:
```
# gammas = [0.1, 0.5, 1.0, 1.1, 1.2, 1.3, 1.5, 2.0]
# hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
# hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'RBF gamm
a', gammas, X_train, y_train, k=3)
```

In [27]:
```
# hybrid_val_scores
```

In [29]:
```
depths = [1, 5, 10, 20, 30, 40, 50]
hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth',
depths, X_train, y_train, k=3)
```

```
max depth: 50                    |████████████████████| 100%
```



In [30]:
```
hybrid_val_scores
```

Out[30]:
```
[0.43631856213201115,
 0.5170953413903522,
 0.5381675446751368,
 0.5399235616155357,
 0.5399235616155356,
 0.5398202665013945,
 0.5398202665013945]
```

In [31]:
```
# clusters = [1, 2, 3, 4, 5, 6]
# hybrid_fn = lambda v: HybridClassifier(n_clusters=v, max_depth=30)
# hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'number of
clusters', clusters, X_train, y_train, k=3)
```

In [ ]:
```
hybrid_val_scores
```

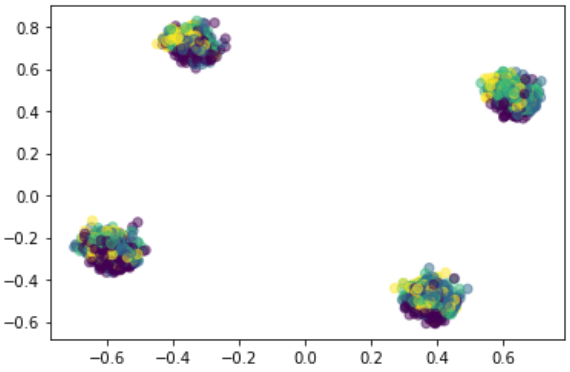
save results for best hybrid classifier

In [51]:
```
clf = HybridClassifier(n_clusters=4, max_depth=30)
clf.fit(X_train, y_train)
```

In [29]:
```
save_predictions(clf, 'hybrid')
```

In [29]:
```
X2_test = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X2_test[:,0], X2_test[:,1], c=clf.predict(X_test), alpha=0.5
)
```

Out[29]:
```
<matplotlib.collections.PathCollection at 0x15cb3b2e0>
```



SVM Model

In [25]:
```
from sklearn.svm import SVC
```

In [40]:
```
# kernels = ['linear', 'poly', 'rbf', 'sigmoid']
# svm_fn = lambda v: SVC(kernel=v)
# svm_train_scores, svm_val_scores = vary_cv(svm_fn, 'kernel', kernels,
 X_train, y_train, k=5)
```

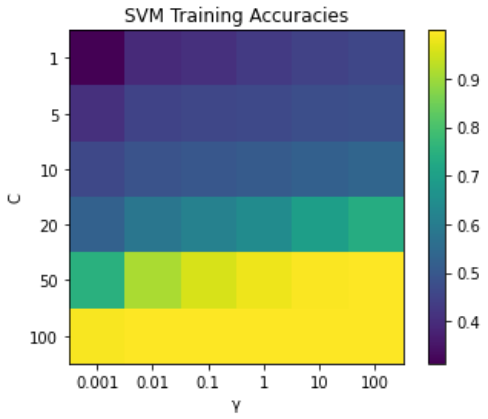
In [41]:
```
# gammas = [0.001, 0.01, 0.1, 1, 10, 100]
# rbf_fn = lambda v: SVC(kernel='rbf', gamma=v)
# rbf_train_scores, rbf_val_scores = vary_cv(rbf_fn, 'gamma', gammas, X_
train, y_train, k=5)
```

In [42]:
```
# C = [1, 5, 10, 20, 50, 100]
# rbf_fn = lambda v: SVC(kernel='rbf', C=v)
# rbf_train_scores, rbf_val_scores = vary_cv(rbf_fn, 'C', C, X_train, y_
train, k=5)
```

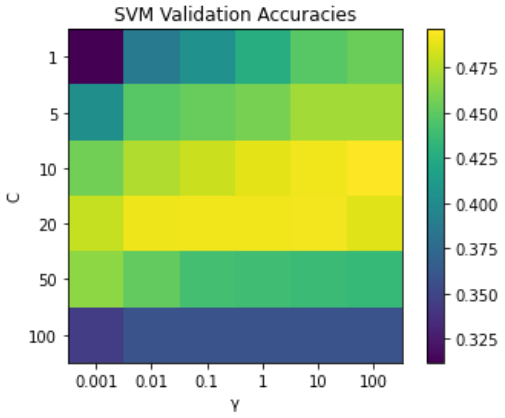
In [33]:
```
gammas = [0.001, 0.01, 0.1, 1, 10, 100]
C = [1, 5, 10, 20, 50, 100]

train_acc = np.zeros((len(gammas), len(C)))
val_acc = np.zeros((len(gammas), len(C)))

for i in range(len(gammas)):
    for j in range(len(C)):
        progress(i*len(C) + j, len(gammas) * len(C), f'γ={gammas[i]}, C={
C[j]}')
        clf = SVC(kernel='rbf', gamma=gammas[i], C=C[j])
        train_acc[i][j], val_acc[i][j] = cv(clf, X_train, y_train, k=3)
```

```
γ=100, C=100                    |████████████████████| 100%
```

In [52]:
```
plt.imshow(train_acc.T)
plt.colorbar()
plt.title('SVM Training Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_train.pdf')
```



In [55]:
```
plt.imshow(test_acc.T)
plt.colorbar()
plt.title('SVM Validation Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_val.pdf')
```

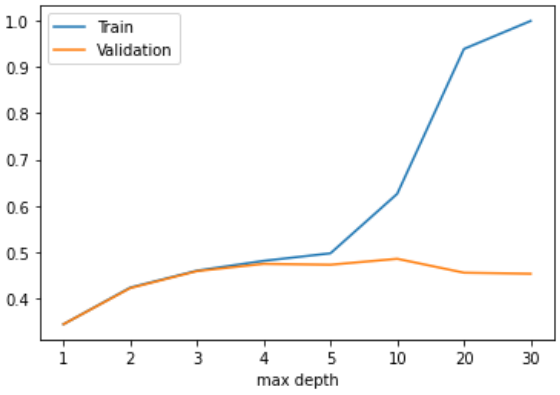


In [49]:
```
########## γ = 10, C = 20
```

Decision Tree

In [22]:
```
from sklearn.tree import DecisionTreeClassifier

depths = [1, 2, 3, 4, 5, 10, 20, 30]
tree_fn = lambda v: DecisionTreeClassifier(max_depth=v, random_state=0)
tree_train_scores, tree_val_scores = vary_cv(tree_fn, 'max depth', depth
s, X_train, y_train, k=10)
```

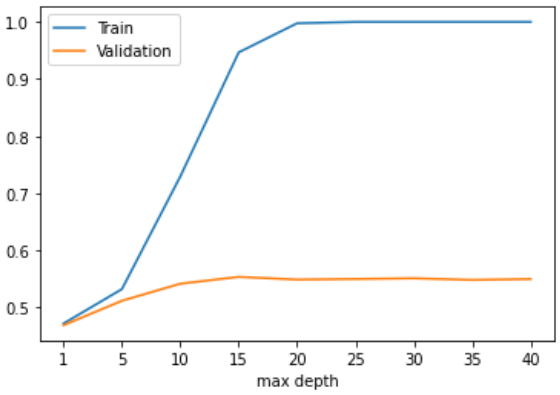
```
max depth: 30                    |████████████████████| 100%
```



Random Forest

In [57]:
```
from sklearn.ensemble import RandomForestClassifier

depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                    |████████████████████| 100%
```



In [ ]:
```
plt.plot(forest_train_scores, label='Train')
plt.plot(forest_val_scores, label='Validation')
plt.xticks(ticks=range(len(forest_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Random Forest')
plt.legend()
plt.savefig('img/training/rf.pdf')
```

In [54]:
```
clf = RandomForestClassifier(max_depth=20, random_state=0)
clf.fit(X_train, y_train)
save_predictions(clf, 'random_forest')
```

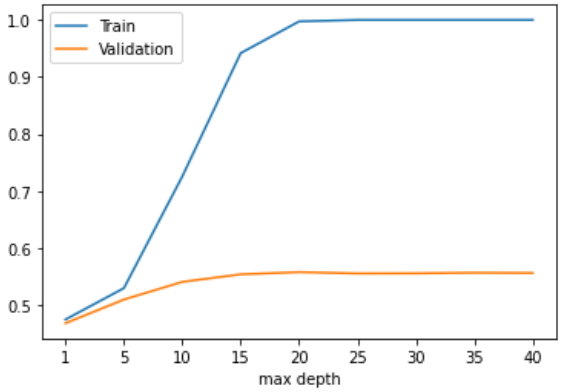
After Correlated Feature Removal

In [62]:
```
for f in ['number_of_reviews', 'require_guest_phone_verification']:
    X_train = X_train.drop(columns=[f])
    X_test = X_test.drop(columns=[f])
```

Random Forest

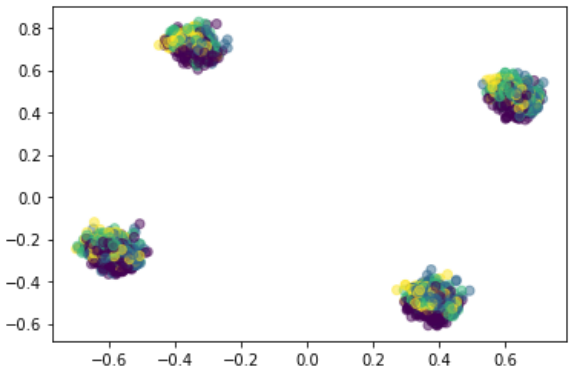
In [63]:
```
depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                    |████████████████████| 100%
```



In [ ]:
```
plt.plot(forest_train_scores, label='Train')
plt.plot(forest_val_scores, label='Validation')
plt.xticks(ticks=range(len(forest_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Random Forest')
plt.legend()
plt.savefig('img/training/rf_removed.pdf')
```

In [33]:
```
clf = RandomForestClassifier(max_depth=20, random_state=0)
clf.fit(X_train, y_train)
save_predictions(clf, 'RF_feature_removal')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/forest.pdf')
```

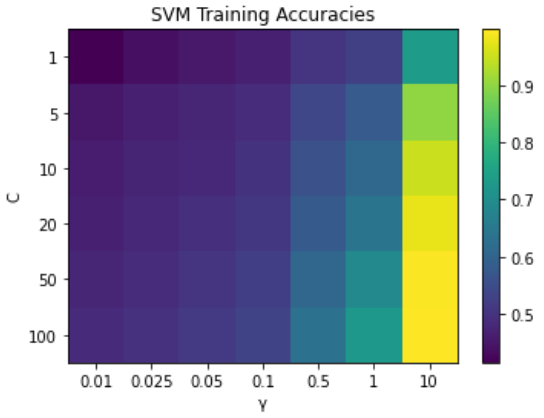


SVM

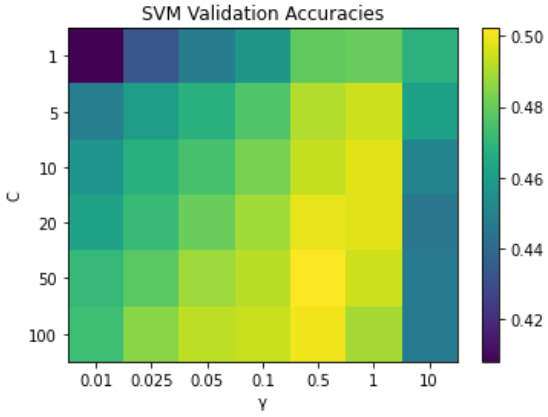
In [35]:
```
gammas = [0.01, 0.025, 0.05, 0.1, 0.5, 1, 10]
C = [1, 5, 10, 20, 50, 100]

train_acc = np.zeros((len(gammas), len(C)))
val_acc = np.zeros((len(gammas), len(C)))

for i in range(len(gammas)):
    for j in range(len(C)):
        progress(i*len(C) + j, len(gammas) * len(C), f'γ={gammas[i]}, C={
C[j]}')
        clf = SVC(kernel='rbf', gamma=gammas[i], C=C[j])
        train_acc[i][j], val_acc[i][j] = cv(clf, X_train, y_train, k=5)
```

γ=10, C=100 |████████████████████| 100%

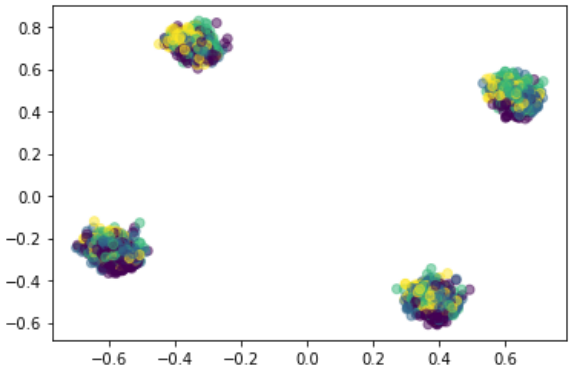
In [36]:
```
plt.imshow(train_acc.T)
plt.colorbar()
plt.title('SVM Training Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_train_removed.pdf')
```



In [37]:
```
plt.imshow(val_acc.T)
plt.colorbar()
plt.title('SVM Validation Accuracies')
plt.xlabel('γ')
plt.ylabel('C')
plt.xticks(ticks=range(len(gammas)), labels=gammas)
plt.yticks(ticks=range(len(C)), labels=C)
plt.savefig('img/training/svm_val_removed.pdf')
```



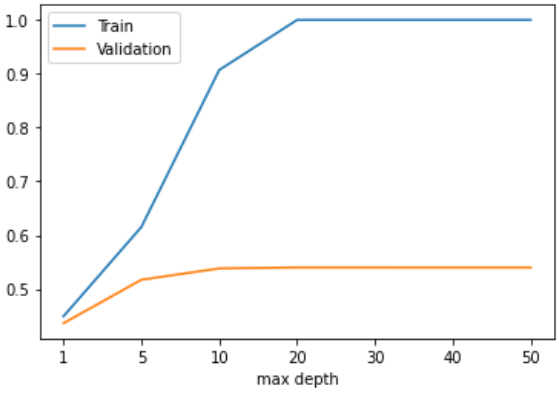
In [32]:
```
clf = SVC(kernel='rbf', gamma=0.5, C=50)
clf.fit(X_train, y_train)
save_predictions(clf, 'svm_removed')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/svm.pdf')
```



Cluster -> Random Forest

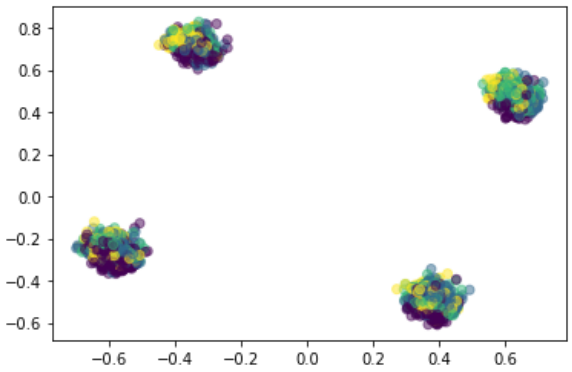
In [20]:
```
depths = [1, 5, 10, 20, 30, 40, 50]
hybrid_fn = lambda v: HybridClassifier(n_clusters=4, max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth',
depths, X_train, y_train, k=3)
```

```
max depth: 50                    |████████████████████| 100%
```



In [ ]:
```
plt.plot(hybrid_train_scores, label='Train')
plt.plot(hybrid_val_scores, label='Validation')
plt.xticks(ticks=range(len(hybrid_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('General Cluster Model')
plt.legend()
plt.savefig('img/training/hybrid_removed.pdf')
```

In [31]:
```
clf = HybridClassifier(n_clusters=4, max_depth=20)
clf.fit(X_train, y_train)
save_predictions(clf, 'hybrid_removed')

X = PCA(n_components=2).fit_transform(X_test)
plt.scatter(X[:,0], X[:,1], c=clf.predict(X_test), alpha=0.5)
plt.savefig('img/preds/hybrid.pdf')
```

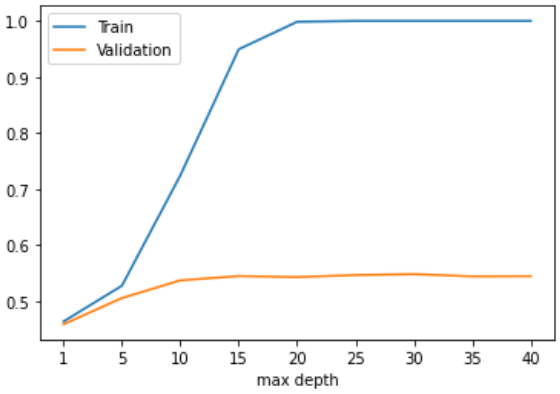


removing other features

In [21]:
```
for f in ['bedrooms']:
    X_train = X_train.drop(columns=[f])
    X_test = X_test.drop(columns=[f])
```

In [22]:
```
depths = [1, 5, 10, 15, 20, 25, 30, 35, 40]
forest_fn = lambda v: RandomForestClassifier(max_depth=v, random_state=0
)
forest_train_scores, forest_val_scores = vary_cv(forest_fn, 'max depth',
depths, X_train, y_train, k=10)
```

```
max depth: 40                    |████████████████████| 100%
```



NOTEBOOK 2: NEIGHBORHOOD CLUSTERS

In [1]:
```
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
from geopy import distance
def dist_from_center(lat, lon):
    buenos_aires_center = (-34.603722, -58.381592)
    return distance.distance(buenos_aires_center, (lat, lon)).km
```

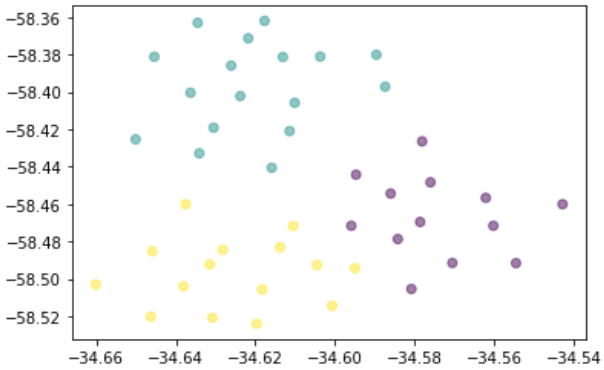
In [3]:
```
coords = {
    'Agronomía': (-34.5950, -58.4943),
    'Almagro': (-34.6114, -58.4210),
    'Balvanera': (-34.6101, -58.4059),
    'Barracas': (-34.6454, -58.3813),
    'Belgrano': (-34.5621, -58.4567),
    'Boedo': (-34.6305, -58.4192),
    'Caballito': (-34.6159, -58.4406),
    'Chacarita': (-34.5860, -58.4544),
    'Coghlan': (-34.5602, -58.4716),
    'Colegiales': (-34.5760, -58.4484),
    'Constitución': (-34.6261, -58.3860),
    'Flores': (-34.6375, -58.4601),
    'Floresta': (-34.6282, -58.4844),
    'La Boca': (-34.6345, -58.3631),
    'La Paternal': (-34.5959, -58.4716),
    'Liniers': (-34.6463, -58.5202),
    'Mataderos': (-34.6601, -58.5031),
    'Monserrat': (-34.6131, -58.3814),
    'Monte Castro': (-34.6183, -58.5057),
    'Nueva Pompeya': (-34.6501, -58.4254),
    'Núñez': (-34.5428, -58.4601),
    'Palermo': (-34.5781, -58.4265),
    'Parque Avellaneda': (-34.6459, -58.4852),
    'Parque Chacabuco': (-34.6341, -58.4329),
    'Parque Chas': (-34.5842, -58.4787),
    'Parque Patricios': (-34.6363, -58.4005),
    'Puerto Madero': (-34.6177, -58.3621),
    'Recoleta': (-34.5874, -58.3973),
    'Retiro': (-34.5896, -58.3802),
    'Saavedra': (-34.5545, -58.4916),
    'San Cristóbal': (-34.6238, -58.4023),
    'San Nicolás': (-34.6037, -58.3812),
    'San Telmo': (-34.6218, -58.3714),
    'Versalles': (-34.6308, -58.5208),
    'Villa Crespo': (-34.5947, -58.4443),
    'Villa Devoto': (-34.6007, -58.5144),
    'Villa General Mitre': (-34.6105, -58.4717),
    'Villa Luro': (-34.6381, -58.5040),
    'Villa Ortúzar': (-34.5786, -58.4696),
    'Villa Pueyrredón': (-34.5808, -58.5054),
    'Villa Real': (-34.6197, -58.5240),
    'Villa Santa Rita': (-34.6138, -58.4832),
    'Villa Urquiza': (-34.5705, -58.4915),
    'Villa del Parque': (-34.6045, -58.4926),
    'Vélez Sársfield': (-34.6315, -58.4923)
}
```

In [4]:
```
dists = {
    nhood: dist_from_center(*coords[nhood]) for nhood in coords
}
```

In [5]:
```
def out(action, bar, percent, done=False):
    done = f'\n' if done is True else ''
    sys.stdout.write('\r%s |%s| ' % (action.ljust(30), bar) + '{0:.0f}%'
.format(percent) + done)

def progress(i, total, action):
    i = i + 1
    ratio = i / total
    percent = 100 * ratio
    filled = int(round(20 * ratio))

    bar = '' * filled + '-' * (20 - filled)

    done = (percent == 100)
    out(action, bar, percent, done)

    if i == total:
        sys.stdout.write('\n')

    sys.stdout.flush()
```

Clustering based on neighborhood

In [6]:
```
from sklearn.cluster import KMeans

X = pd.DataFrame.from_dict(coords, orient='index')

n_clusters = 3
neighborhood_kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(
X)

X_np = X.to_numpy()
plt.scatter(X_np[:,0], X_np[:,1], c=neighborhood_kmeans.labels_, alpha=
0.5)

plt.savefig('img/clusters/nhoods.pdf')
```



In [7]:
```
def cluster_from_nhood(nhood):
    c = coords[nhood]
    return neighborhood_kmeans.predict([list(c)]).item()
```

Import data

In [8]:
```
X_train = pd.read_csv('data/train.csv')
y_train = X_train['price']
X_test = pd.read_csv('data/test.csv')

X_train = X_train.drop(columns=['price'])

features = X_train.columns
labels = list(np.unique(y_train))

string_features = {
    'neighbourhood': ['Agronomía', 'Almagro', 'Balvanera', 'Barracas',
'Belgrano',
       'Boedo', 'Caballito', 'Chacarita', 'Coghlan', 'Colegiales',
       'Constitución', 'Flores', 'Floresta', 'La Boca', 'La Paternal',
       'Liniers', 'Mataderos', 'Monserrat', 'Monte Castro',
       'Nueva Pompeya', 'Núñez', 'Palermo', 'Parque Avellaneda',
       'Parque Chacabuco', 'Parque Chas', 'Parque Patricios',
       'Puerto Madero', 'Recoleta', 'Retiro', 'Saavedra', 'San Cristóba
l',
       'San Nicolás', 'San Telmo', 'Versalles', 'Villa Crespo',
       'Villa Devoto', 'Villa General Mitre', 'Villa Luro',
       'Villa Ortúzar', 'Villa Pueyrredón', 'Villa Real',
       'Villa Santa Rita', 'Villa Urquiza', 'Villa del Parque',
       'Vélez Sársfield'],
    'room_type': ['Entire home/apt', 'Hotel room', 'Private room', 'Shar
ed room'],
    'host_is_superhost': ['f', 't'],
    'bed_type': ['Airbed', 'Couch', 'Futon', 'Pull-out Sofa', 'Real Bed'
],
    'instant_bookable': ['f', 't'],
    'is_business_travel_ready': ['f'],
    'cancellation_policy': ['flexible', 'moderate', 'strict_14_with_grac
e_period', 'super_strict_30', 'super_strict_60'],
    'require_guest_profile_picture': ['f', 't'],
    'require_guest_phone_verification': ['f', 't'],
}
```

Add in distance from center of city

In [9]:
```
# set new "distance" column to be all 0s
X_train['distance'] = 0.0
X_test['distance'] = 0.0

# fill in the distances from the city center
for i in range(X_train.shape[0]):
    X_train.at[i, 'distance'] = dists[X_train.at[i, 'neighbourhood']]
for i in range(X_test.shape[0]):
    X_test.at[i, 'distance'] = dists[X_test.at[i, 'neighbourhood']]
```

save IDs

In [10]:
```
train_ids = X_train['id']
test_ids = X_test['id']
```

prediction saving function

In [11]:
```
def save_predictions(clf, filename):
    y_pred = clf.predict(X_test)
    assert(y_pred.shape[0] == 4149)

    y_pred = pd.DataFrame(data=y_pred, columns=['price'])
    y_pred['id'] = test_ids
    y_pred = y_pred.reindex(['id','price'], axis=1).astype(int)

    y_pred.to_csv(f'results/{filename}.csv', index=False)
```

Data Preprocessing

In [12]:
```
from datetime import date

def data2float(X):
    X = X.copy()

    # turn strings into integers
    for f in string_features:
        for i in range(X.shape[0]):
            X.at[i, f] = string_features[f].index(X.at[i, f])

    # turn dates into days relative to today
    today = date.today()
    for f in ['last_review', 'host_since']:
        for i in range(X.shape[0]):
            m, d, y = (int(x) for x in X.at[i, f].split('/'))
            X.at[i, f] = (today - date(y, m, d)).days

    return X.astype(float)
```

In [13]:
```
from sklearn.preprocessing import MinMaxScaler

def normalize_df(df):
    min_max_scaler = MinMaxScaler()
    return pd.DataFrame(min_max_scaler.fit_transform(df.values), columns
=df.columns)
```

In [14]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.ensemble import RandomForestClassifier


class HybridClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, max_depth):
        self.n_clusters = 3
        self.max_depth = max_depth
        self.cluster_models = None

    def _get_data_clusters(self, X, y):
        cluster_data = {i: () for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            idx = [i for i in range(len(X)) if cluster_from_nhood(X.at[i
, 'neighbourhood']) == c]
            X_c = normalize_df(data2float(X.loc[idx].reset_index(drop=Tr
ue)))
            y_c = y.loc[idx].reset_index(drop=True)
            cluster_data[c] = (X_c, y_c)
        return cluster_data

    def fit(self, X, y):
        # divide data into clusters based on neighborhood
        cluster_data = self._get_data_clusters(X, y)

        # fit a model for each cluster
        self.cluster_models = {i: None for i in range(self.n_clusters)}
        for c in range(self.n_clusters):
            X, y = cluster_data[c]
            clf = RandomForestClassifier(max_depth=self.max_depth, rando
m_state=0)
            clf.fit(X, y)
            self.cluster_models[c] = clf

    def predict(self, X):
        '''
        Assume X is **not** normalized or anything yet
        '''
        if self.cluster_models is None:
            raise AssertionError('You have to train the model first, du
h')

        pred_clusters = [cluster_from_nhood(n) for n in X['neighbourhoo
d']]
        X = normalize_df(data2float(X))

#         pred_y = [self.cluster_models[pred_clusters[i]].predict([X.loc
[i]]) for i in range(len(X))]
        pred_y = []
        for i in range(len(X)):
            x = X.loc[i]
            c = pred_clusters[i]
            pred_y.append(self.cluster_models[c].predict([x]))
        pred_y = np.array(pred_y).flatten()
        return pred_y
```

```
    def score(self, X, y):
        pred_y = self.predict(X)
        num_correct = sum([1 if pred_y[i] == y.loc[i] else 0 for i in ra
nge(len(X))])
        return num_correct / len(X)
```

In [15]:
```
# clf = HybridClassifier(max_depth=10)
# clf.fit(X_train, y_train)
```

In [16]:
```
# clusters_test = [cluster_from_nhood(n) for n in X_test['neighbourhoo
d']]
# X_test = normalize_df(data2float(X_test))
```

Cross-Validation

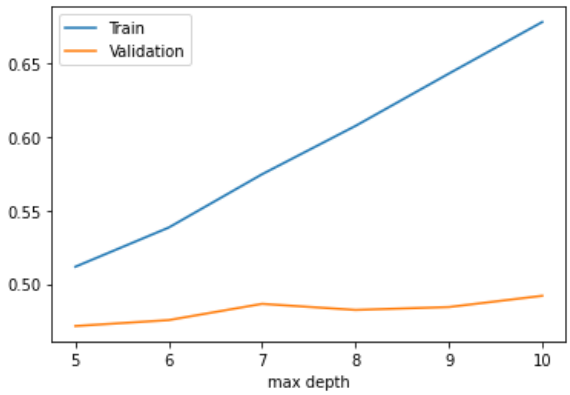
In [15]:
```
from sklearn.model_selection import KFold
from sklearn.base import clone

def cv(clf, X, y, k):
    X = X.copy()
    y = y.copy()
    base_clf = clone(clf)

    train_scores = []
    val_scores = []

    kf = KFold(n_splits=k)
    kf.get_n_splits(X)

    for train_index, val_index in kf.split(X):
        # split data into training and validation
        X_train, X_val = X.loc[train_index], X.loc[val_index]
        y_train, y_val = y.loc[train_index], y.loc[val_index]

        # reset indices for the data
        X_train = X_train.reset_index(drop=True)
        X_val = X_val.reset_index(drop=True)
        y_train = y_train.reset_index(drop=True)
        y_val = y_val.reset_index(drop=True)

        # train and score a classifier
        clf = clone(base_clf)
        clf.fit(X_train, y_train)
        train_scores.append(clf.score(X_train, y_train))
        val_scores.append(clf.score(X_val, y_val))

    train_mean = sum(train_scores) / len(train_scores)
    val_mean = sum(val_scores) / len(val_scores)
    return train_mean, val_mean
```

In [16]:
```
def vary_cv(clf_fn, prop, values, X, y, k):
    train_scores = []
    val_scores = []

    i = 0
    for v in values:
        progress(i, len(values), f'{prop}: {v}')
        i += 1

        clf = clf_fn(v)
        train_mean, val_mean = cv(clf, X, y, k=k)
        train_scores.append(train_mean)
        val_scores.append(val_mean)

    plt.plot(train_scores, label='Train')
    plt.plot(val_scores, label='Validation')
    plt.xticks(ticks=range(len(train_scores)), labels=values)
    plt.xlabel(prop)
    plt.legend()
    plt.show()

    return train_scores, val_scores
```

Training

In [17]:
```
depths = [5, 6, 7, 8, 9, 10]
hybrid_fn = lambda v: HybridClassifier(max_depth=v)
hybrid_train_scores, hybrid_val_scores = vary_cv(hybrid_fn, 'max depth',
depths, X_train, y_train, k=3)
```

```
max depth: 10                    |████████████████████| 100%
```



In [ ]:
```
plt.plot(hybrid_train_scores, label='Train')
plt.plot(hybrid_val_scores, label='Validation')
plt.xticks(ticks=range(len(hybrid_train_scores)), labels=depths)
plt.xlabel('max depth')
plt.ylabel('Accuracy')
plt.title('Neighborhood Cluster Model')
plt.legend()
plt.savefig('img/training/nhood_removed.pdf')
```

In [21]:
```
clf = HybridClassifier(max_depth=10)
clf.fit(X_train, y_train)
save_predictions(clf, 'nhood_clusters_removed')
```

References

1. Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.
2. F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.