W4995 Applied Machine Learning
Fall 2021

Lecture 7

Dr. Vijay Pappu

Announcements

- Midterm
- Project deliverable 1 grades released
- Project deliverable 2 due on 11/10
- HW3 will be posted on 11/10

In today’s lecture, we will cover...

- Learning with imbalance data
- Learning with sparse data

Learning with Imbalance Data

Imbalance data

- Imbalanced data is generally discussed in the context of classification tasks
- Imbalance is generally defined in terms of classes
- Generally two sources:
  - Imbalance due to different cost (asymmetric cost)
  - Imbalance due to samples (asymmetric data)
- Asymmetric cost is generally applicable when getting one class wrong is more costlier than the other class (cancer detection, fraud detection)

Imbalance data - Why is this important?

- Imbalanced datasets are more the norm!
- The cost is never symmetric!

Imbalance data - Changing thresholds

```
data = load_breast_cancer()
X_dev, X_test, y_dev, y_test = train_test_split(data.data, data.target,
                                                stratify=data.target, test_size=0.2,
                                                random_state=42)
lr = LogisticRegression().fit(X_dev, y_dev)
print(classification_report(y_test, lr.predict(X_test)))
```

```
              precision    recall  f1-score   support

           0       0.97      0.90      0.94        42
           1       0.95      0.99      0.97        72

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```

Imbalance data - Changing thresholds

```
y_pred = lr.predict_proba(X_test)[:,1] > 0.9
print(classification_report(y_test, y_pred))
```

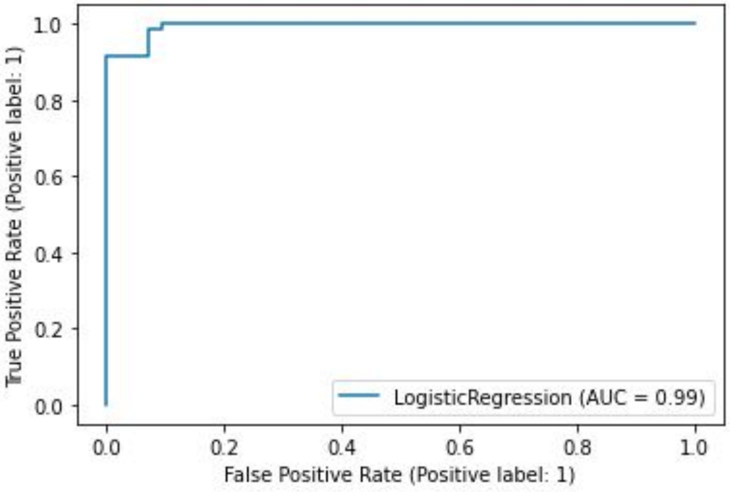
```
              precision    recall  f1-score   support

           0       0.79      1.00      0.88        42
           1       1.00      0.85      0.92        72

    accuracy                           0.90       114
   macro avg       0.90      0.92      0.90       114
weighted avg       0.92      0.90      0.91       114
```

Imbalance data - How can we decide thresholds?

```
y_score = lr.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_score)
print(thresholds)
plot_roc_curve(lr, X_test, y_test)
```

```
[1.99992468e+00 9.99924683e-01 8.39939594e-01 7.91870270e-01
 5.35375254e-01 5.23544831e-01 4.08154789e-01 2.08671490e-50]
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x123441150>
```



Training Classifiers with Imbalance Data

- Change data
  - Random Undersampling
  - Random Oversampling
  - Ensemble Resampling
  - Synthetic Minority Oversampling Technique (SMOTE)
- Change training procedure
  - Class weights

Sampling in Python

- Sklearn pipelines does not support resampling
- We will use imbalanced-learn package
    - Compatible with sklearn API

Training Classifiers with Imbalance Data - Example

```
data = fetch_openml("mammography", as_frame=True)
X, y = data.data, data.target
print(X.shape)
y.value_counts()
```

```
(11183, 6)

-1    10923
1       260
```

```
y.value_counts(normalize=True)
```

```
-1     0.97675
1      0.02325
Name: class, dtype: float64
```

```
X_dev, X_test, y_dev, y_test = train_test_split(data.data, data.target == '1',
                                                stratify=data.target, test_size=0.2,
                                                random_state=42)
```

```
y_dev.value_counts(normalize=True)
```

```
False    0.976749
True     0.023251
Name: class, dtype: float64
```

```
y_test.value_counts(normalize=True)
```

```
False    0.976755
True     0.023245
Name: class, dtype: float64
```

Training Classifiers with Imbalance Data - Example

```
scores = cross_validate(LogisticRegression(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9047407909496835, 0.6045937994904841)
```

Training Classifiers with Imbalance Data - Example

```
scores = cross_validate(RandomForestClassifier(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.936680622193963, 0.7306694006997267)
```

Training Classifiers with Imbalance Data - Example

```
scores = cross_validate(HistGradientBoostingClassifier(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9303104997165956, 0.7096473396572363)
```

Training Classifiers with Imbalance Data - Random Undersampling

```
rus = RandomUnderSampler(replacement=False)
X_dev_subsample, y_dev_subsample = rus.fit_resample(X_dev, y_dev)
print(X_dev.shape)
print(X_dev_subsample.shape)
y_dev_subsample.value_counts()
```

```
(8946, 6)
(416, 6)

False    208
True     208
Name: class, dtype: int64
```

Training Classifiers with Imbalance Data - Random Undersampling

```
scores = cross_validate(LogisticRegression(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9047407909496835, 0.6045937994904841)
```

```
rus = RandomUnderSampler(replacement=False, random_state=42)
sample_pipe = imb_make_pipeline(rus, LogisticRegression())
scores = cross_validate(sample_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9066341667518565, 0.5874283673914457)
```

Training Classifiers with Imbalance Data - Random Undersampling

```
scores = cross_validate(RandomForestClassifier(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.936680622193963, 0.7306694006997267)
```

```
rus = RandomUnderSampler(replacement=False)
sample_pipe_rf = imb_make_pipeline(rus, RandomForestClassifier())
scores = cross_validate(sample_pipe_rf, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9399390167297821, 0.6220152272801691)
```

Training Classifiers with Imbalance Data - Random Oversampling

```
ros = RandomOverSampler()
X_dev_oversample, y_dev_oversample = ros.fit_resample(X_dev, y_dev)
print(X_dev.shape)
print(X_dev_oversample.shape)
y_dev_oversample.value_counts()
```

```
(8946, 6)
(17476, 6)

False    8738
True     8738
Name: class, dtype: int64
```

Training Classifiers with Imbalance Data - Random Oversampling

```
X_dev_oversample.value_counts()
```

```
attr1      attr2      attr3      attr4      attr5      attr6
-0.784415  -0.470195  -0.591631  -0.859553  -0.377866  -0.945723   2900
-0.106883  -0.324216  -0.140816   1.266203  -0.377866   1.501103     57
-0.011182  -0.417112  -0.185897   1.352280   1.936850   1.443208     56
 1.806975   1.042677  -0.546550   3.001660  13.750423   0.690573     56
 1.761411  -0.218050  -0.546550   1.788148   8.215644   1.205534     56
                                                                   ...
-0.062166  -0.377300   0.580489  -0.859553  -0.377866  -0.945723      1
-0.062335  -0.063224  -0.456387   0.873373  -0.377866   1.476726      1
           -0.120731  -0.321142   0.580965   3.019336   1.263429      1
-0.062505  -0.222474   0.535408   0.758182  -0.377866   0.788080      1
 31.508443  3.559706  -0.591631  -0.859553  -0.377866  -0.945723      1
Length: 6299, dtype: int64
```

Training Classifiers with Imbalance Data - Random Oversampling

```
scores = cross_validate(LogisticRegression(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9047407909496835, 0.6045937994904841)
```

```
rus = RandomOverSampler()
oversample_pipe = imb_make_pipeline(ros, LogisticRegression())
scores = cross_validate(oversample_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9120860887402029, 0.5263717213991833)
```

Training Classifiers with Imbalance Data - Random Oversampling

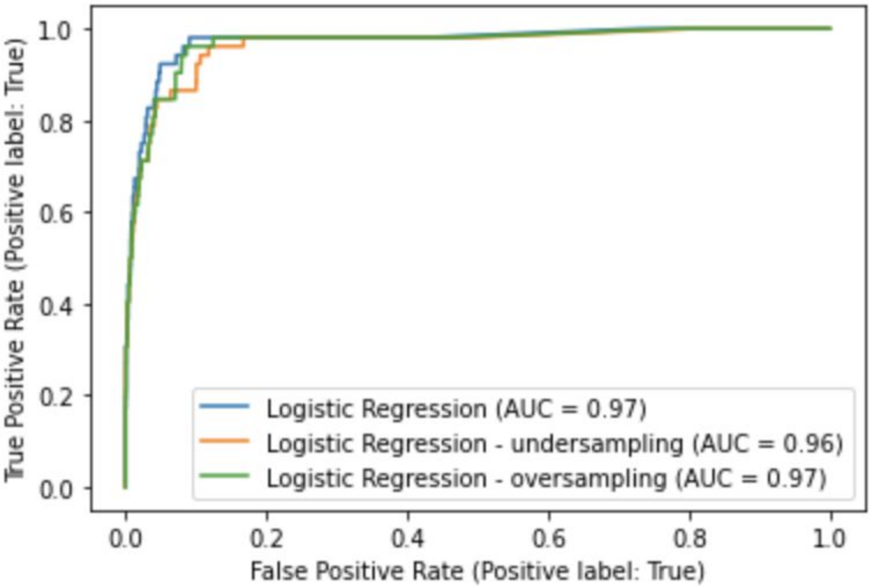
```
scores = cross_validate(RandomForestClassifier(),
                        X_dev, y_dev, cv=10,
                        scoring = ['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

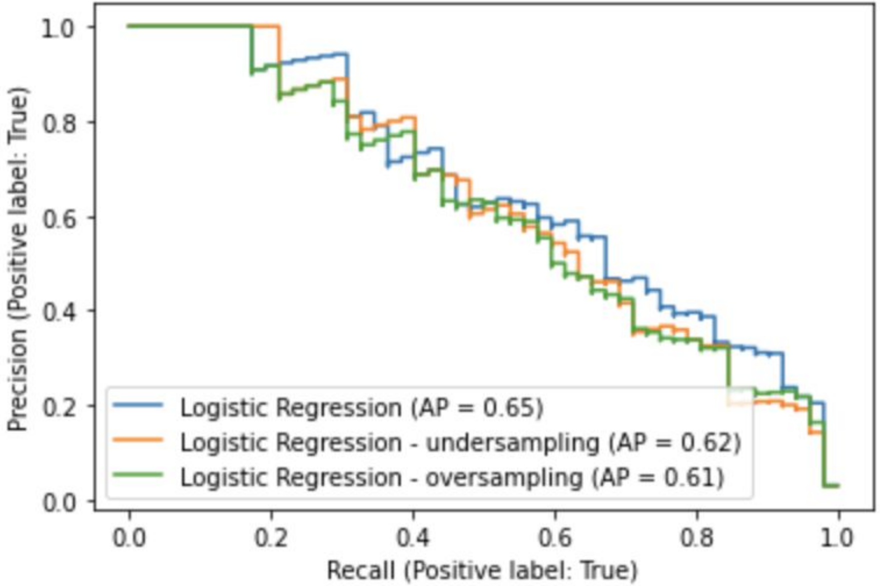
```
(0.9366806221963, 0.7306694006997267)
```

```
rus = RandomOverSampler()
oversample_pipe = imb_make_pipeline(ros, RandomForestClassifier())
scores = cross_validate(oversample_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

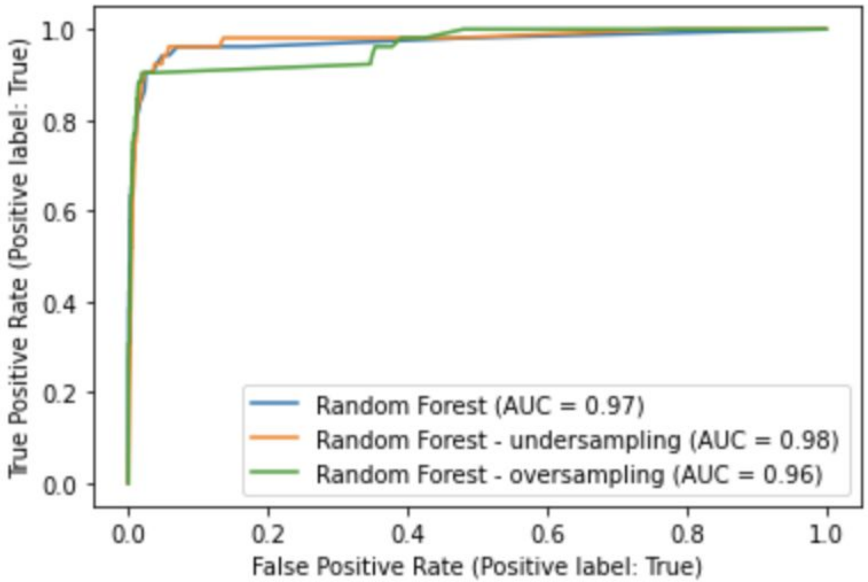
```
(0.9089298044965494, 0.6999995535829469)
```

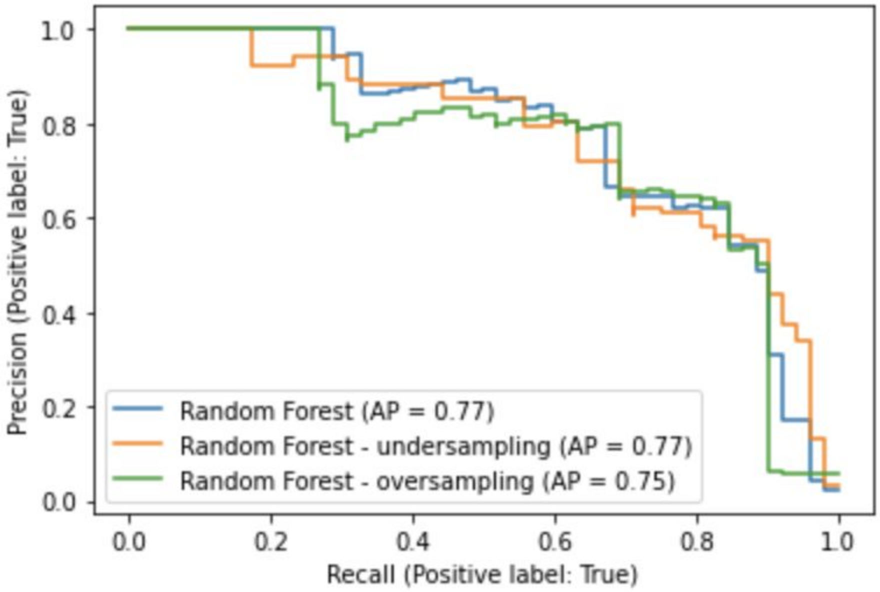
Visualization for Logistic Regression





Visualization for Random Forests





Training Classifiers with Imbalance Data - Ensemble Resampling

- Ensemble methods offer a better way to leverage data from majority class
- A random re-sample of majority class is used for training each instance in an ensemble
- The minority class is retained while training the instance.
- Higher number of samples from majority class are utilized as compared to under-sampling
- Currently available in [imbalanced-learn] package

Ensemble Resampling - Example

```
rus = RandomUnderSampler(replacement=False, random_state=42)
tree = DecisionTreeClassifier(max_features='auto', random_state=42)
under_sample_bagging = imb_make_pipeline(rus, BaggingClassifier(base_estimator=tree,
                                                                 random_state=42))
scores = cross_validate(under_sample_bagging, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.938411825919198, 0.458717954029249)
```

```
tree = DecisionTreeClassifier(max_features='auto')
resampled_bagging = BalancedBaggingClassifier(tree, random_state=42)
scores = cross_validate(resampled_bagging,
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9418277801119164, 0.5248815292096259)
```

Ensemble Resampling - Example

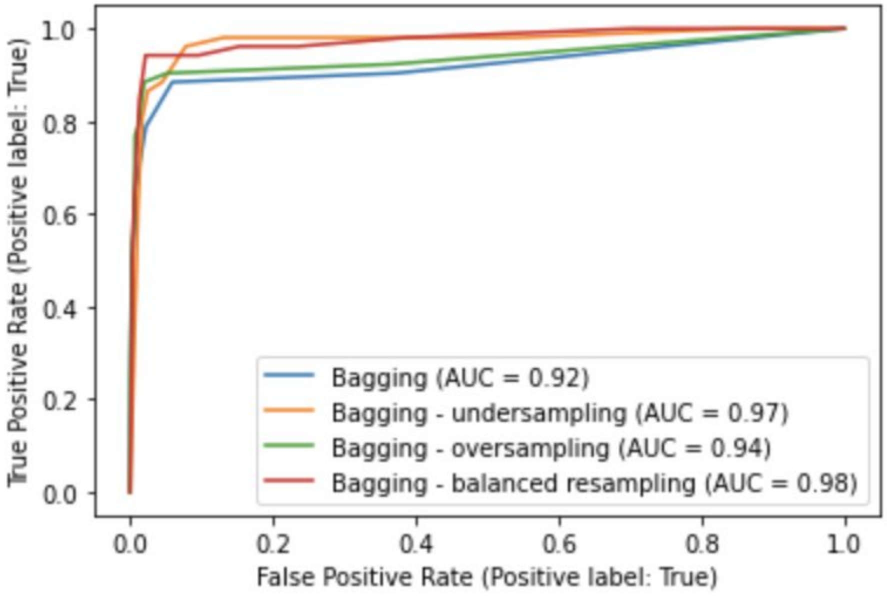
```
rus = RandomUnderSampler(replacement=False, random_state=42)
under_sample_rf = imb_make_pipeline(rus, RandomForestClassifier(random_state=42))
scores = cross_validate(under_sample_rf,
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

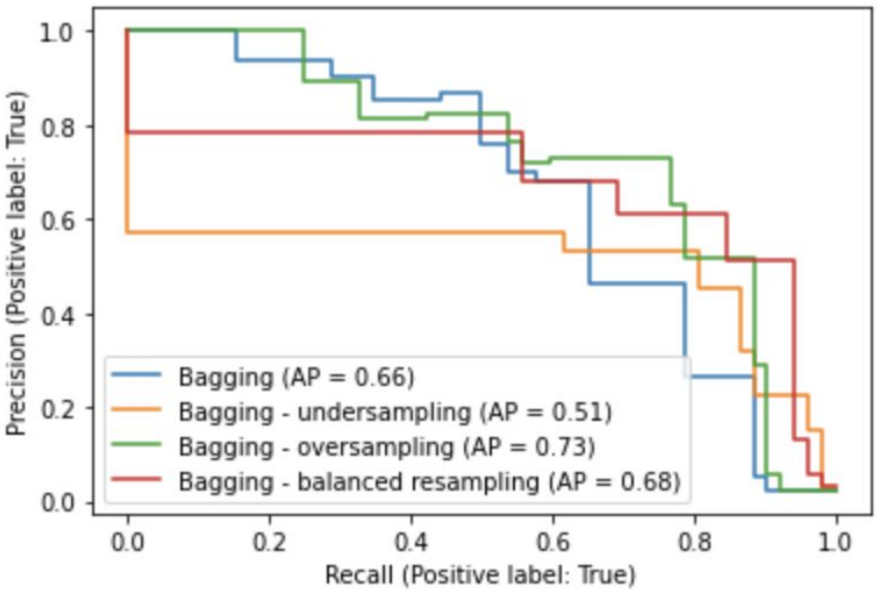
```
(0.9430329687084388, 0.6189073324753068)
```

```
resampled_rf = BalancedRandomForestClassifier(random_state=42)
scores = cross_validate(resampled_rf,
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9439221558240938, 0.6380276519411647)
```

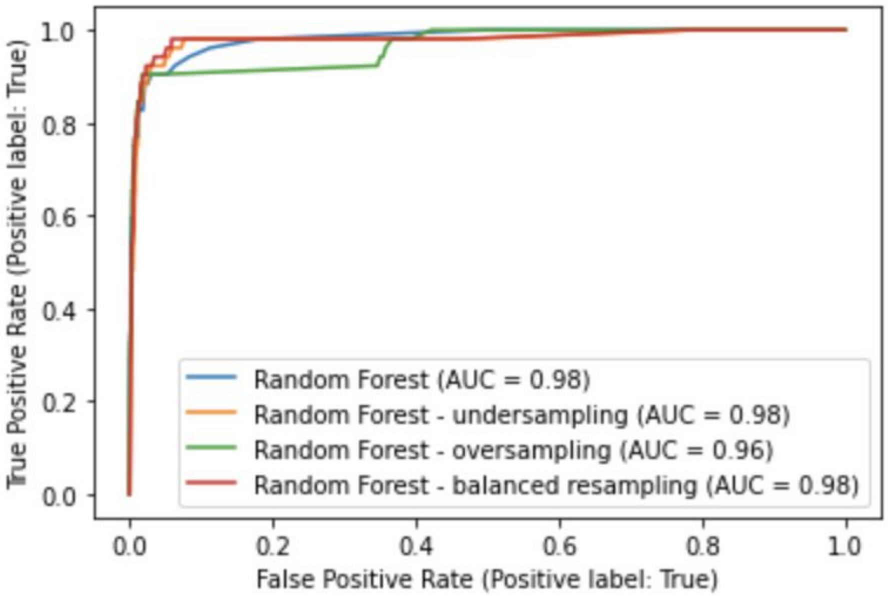
Ensemble Resampling - Example

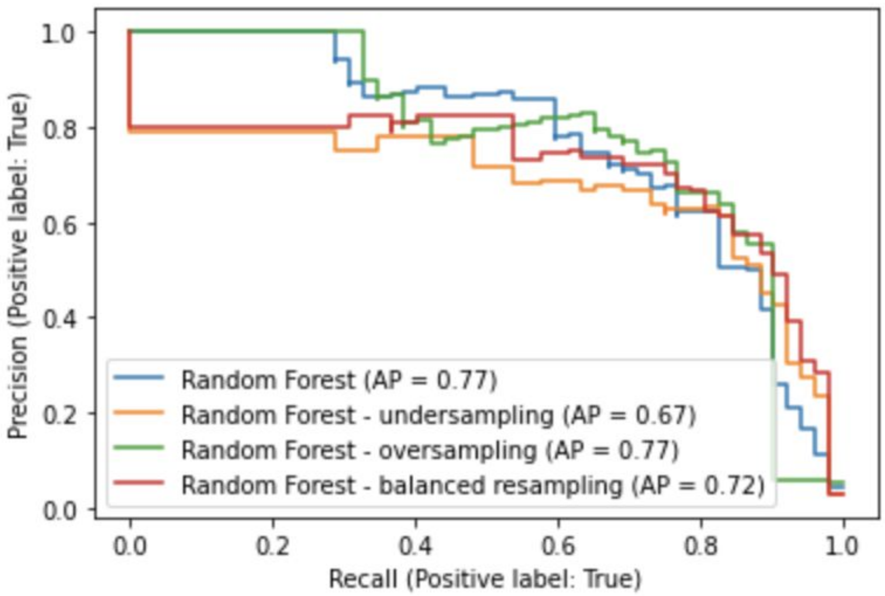




Bagging

Ensemble Resampling - Example



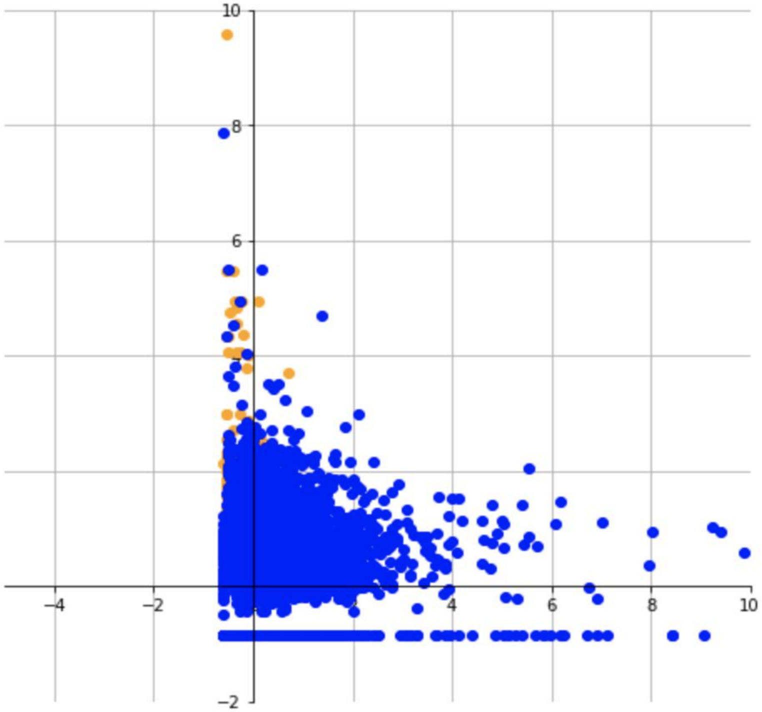


Random Forests

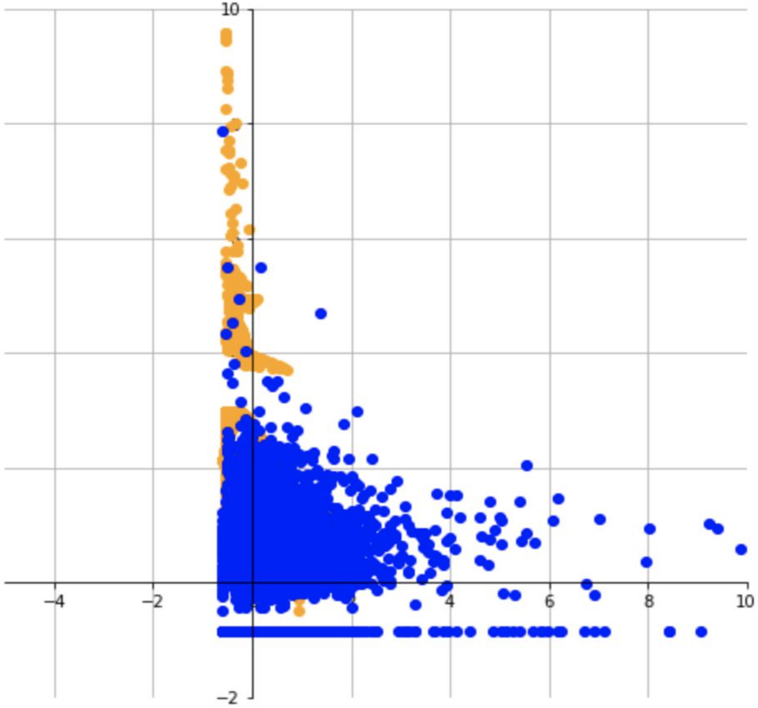
Training Classifiers with Imbalance Data - SMOTE

- Synthetic Minority Oversampling Technique (SMOTE) is a popular method to handle training with imbalanced datasets
- SMOTE adds synthetic interpolated samples to minority class
- The following procedure is repeated for every original data point in minority class:
  - Pick a neighbor from *k* nearest neighbors
  - Sample a point randomly from the line joining the two data points.
  - Add the point to the minority class
- Leads to large datasets (due to oversampling)

Training Classifiers with Imbalance Data - SMOTE



original dataset



SMOTE-sampled dataset

SMOTE - Example

```
ros = RandomOverSampler(random_state=42)
oversample_lr_pipe = imb_make_pipeline(ros, LogisticRegression())
scores = cross_validate(oversample_lr_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9130398683034094, 0.5287701141476036)
```

```
smote = SMOTE(random_state=42)
smote_lr_pipe = imb_make_pipeline(smote, LogisticRegression())
scores = cross_validate(smote_lr_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.914084160017845, 0.5180001398437424)
```

SMOTE - Example

```
ros = RandomOverSampler(random_state=42)
oversample_rf_pipe = imb_make_pipeline(ros, RandomForestClassifier())
scores = cross_validate(oversample_rf_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

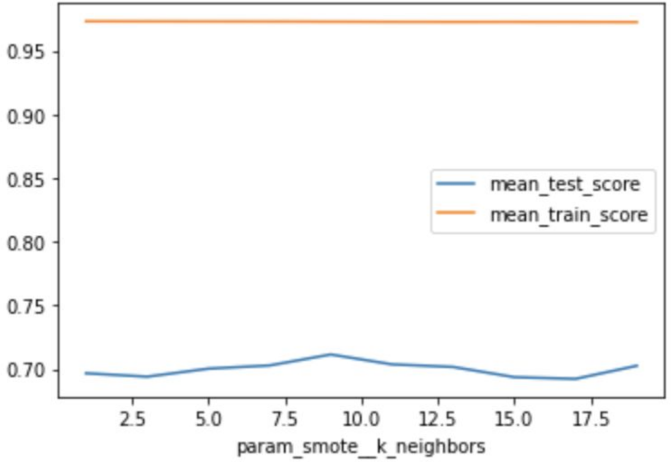
```
(0.9133267794904363, 0.6897708904817222)
```

```
smote = SMOTE(random_state=42)
smote_rf_pipe = imb_make_pipeline(smote, RandomForestClassifier())
scores = cross_validate(smote_rf_pipe, X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

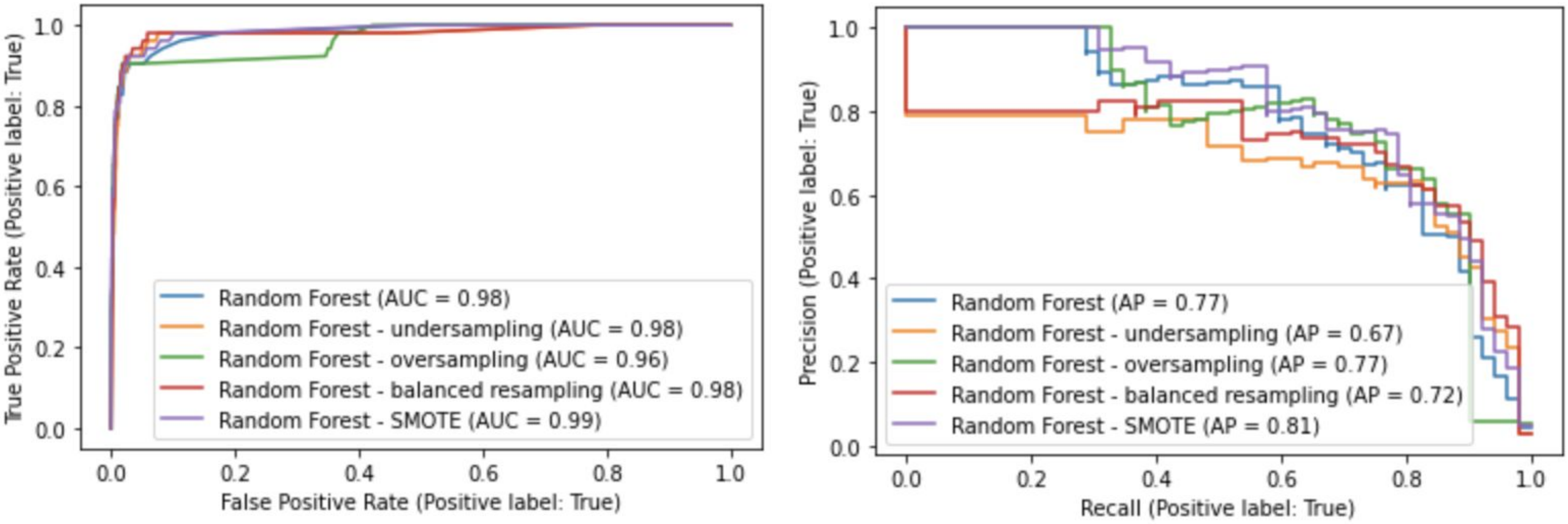
```
(0.9337893077669023, 0.7025735766184975)
```

SMOTE - Example

```
param_grid = {'smote__k_neighbors': np.arange(1, 20, 2)}
gs = GridSearchCV(smote_rf_pipe, param_grid,
                  cv=10, scoring='average_precision',
                  return_train_score=True)
gs.fit(X_dev, y_dev)
results = pd.DataFrame(gs.cv_results_)
results.plot('param_smote__k_neighbors', ['mean_test_score',
                                          'mean_train_score'])
```



SMOTE - Example



Training Classifiers with Imbalance Data - Class Weights

- Oversampling/Undersampling changes the dataset
    - expensive to train (in case of oversampling)
    - Not leverage all data (in case of undersampling)
- Reweight each sample during training
- Modify the loss function to account for class weights
- Works for most models
- Similar effect as oversampling (except that this is not random)

Adding Class Weights - Linear Models

$$\underset{w,\ b}{Min}\left(-\sum_{i=1}^{m} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right) + \alpha \| w \|_2^2$$

Logistic regression (without class weights)

$$\underset{w,\ b}{Min}\left(-\sum_{i=1}^{m} c_{y_i}\left(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right)\right) + \alpha \| w \|_2^2$$

Logistic regression (with class weights)

Adding Class Weights - Linear Models

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m}\left(Max\left(0,\ 1-y_i\left(w^T x_i + b\right)\right)\right) + \frac{1}{2}\| \mathbf{w} \|_2^2$$

Soft-margin SVMs (without class weights)

$$\underset{w,b}{Min}\ C\sum_{i=1}^{m} c_{y_i}\left(Max\left(0,\ 1-y_i\left(w^T x_i + b\right)\right)\right) + \frac{1}{2}\| \mathbf{w} \|_2^2$$

Soft-margin SVMs (with class weights)

Adding Class Weights - Tree-based Models

$$Entropy(node_m) = -\sum_{i=1}^{K} p_{im} \log_2 p_{im}$$

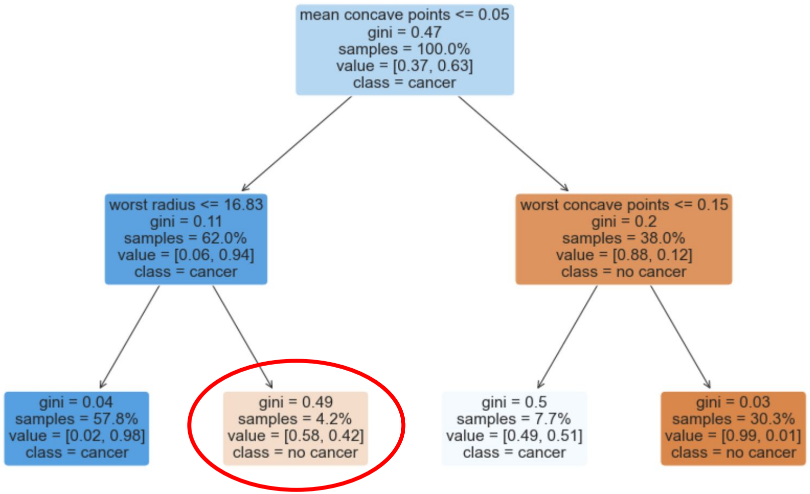
$$Gini\ Index(node_m) = 1 - \sum_{i=1}^{K} p_{im}^2$$

$$Entropy(node_m) = -\sum_{i=1}^{K} c_i p_{im} \log_2 p_{im}$$

$$Gini\ Index(node_m) = 1 - \sum_{i=1}^{K} c_i p_{im}^2$$

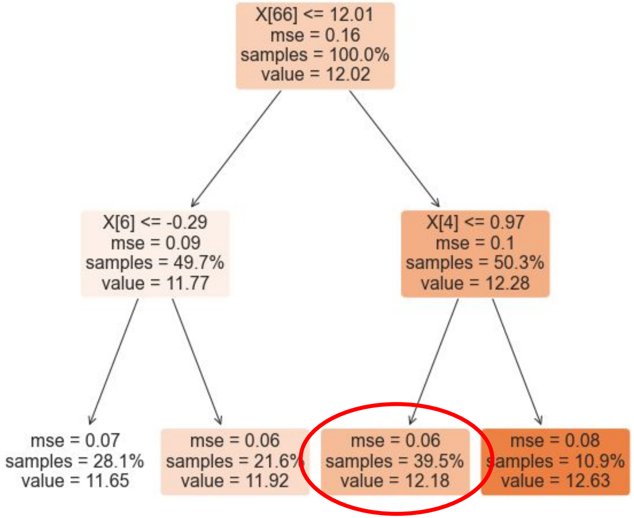
Adding Class Weights - Tree-based Models

Classification Trees



weighted majority voting

Regression Trees



weighted sample mean

Adding Class Weights - Example

```
scores = cross_validate(LogisticRegressionCV(),
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9046980053787539, 0.6063330397221185)
```

```
scores = cross_validate(LogisticRegressionCV(class_weight='balanced'),
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9067204927441367, 0.5724649513533765)
```

Adding Class Weights - Example

```
scores = cross_validate(RandomForestClassifier(),
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9301318665020037, 0.7251947309333016)
```

```
scores = cross_validate(RandomForestClassifier(class_weight='balanced'),
                        X_dev, y_dev, cv=10,
                        scoring=['roc_auc', 'average_precision'])
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

```
(0.9092748344852369, 0.6947892294049234)
```

Training Classifiers with Imbalance Data - Practical Considerations

- Several techniques exist to handle imbalance data (undersampling, oversampling, SMOTE, class weighting, ensemble resampling etc.)
- SMOTE adds synthetic interpolated data to minority class
- Undersampling uses only subset of data, while oversampling could be expensive to train
- Ensemble resampling leverages majority class “smartly”
- SMOTE and ensemble resampling techniques tend to work well in practice

Questions?

Let’s take a 10 min break!

Learning with Sparse Data

Learning with Sparse Data

- Features with sparse data have most values equal to *zero*
- Sparsity for a feature is defined as the ratio of non-zero entries to total number of entries
- Sparse data is also ubiquitous
- Some preprocessing techniques naturally lead to sparse data
- Typically appears in NLP and recommender tasks

Sparse Data v.s. Missing Data

- In sparse data, all values are known while all values are not known in missing data

user_id	Feature 1	Feature 2
123	1	0.1
456	0	null
789	0	0
135	2	null
246	0	0.3

Learning with Sparse Data - Problems

- Increases time and space complexity for models
- Some model algorithms & diagnostic measures perform poorly on sparse data
- Models trained with sparse data could overfit and thus not generalize
- Models could underestimate the importance of sparse features

Learning with Sparse Data - Example

```
data = pd.read_csv('ml-100k/u.data', sep="\t", header=None)
data.columns = ["user_id", "item_id", "rating", "timestamp"]
data.drop(["timestamp"], axis=1, inplace=True)
display(data.head())
display(data.shape)
```

	user_id	item_id	rating
0	196	242	3
1	186	302	3
2	22	377	1
3	244	51	2
4	166	346	1

```
(100000, 3)
```

https://grouplens.org/datasets/movielens/100k/

Learning with Sparse Data - Example

```
print(f"# of unique user entries: ", len(data['user_id'].unique()))
print(f"# of unique item entries: ", len(data['item_id'].unique()))
print(f"# of unique entries:", np.sum(len(data['user_id'].unique())
                                      + len(data['item_id'].unique())))
print(f"Rating distribution: \n", data["rating"].value_counts().sort_index())
```

```
# of unique user entries:  943
# of unique item entries:  1682
# of unique entries: 2625
Rating distribution:
 1     6110
2    11370
3    27145
4    34174
5    21201
Name: rating, dtype: int64
```

Learning with Sparse Data - Example

```
data_processed = pd.get_dummies(data, columns=['user_id', 'item_id'])
data_processed["rating"] = data_processed["rating"] >=4
display(data_processed["rating"].value_counts())
display(data_processed.shape)
```

```
True     55375
False    44625
Name: rating, dtype: int64
(100000, 2626)
```

Learning with Sparse Data - Example

```
display(data_processed.head())
```

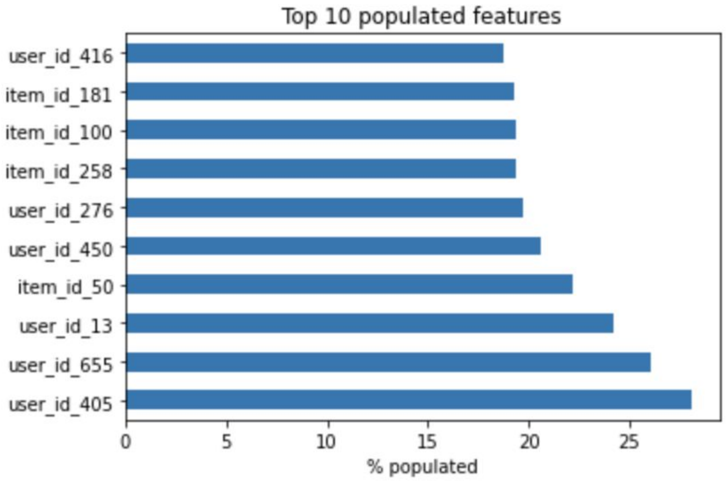
	rating	user_id_1	user_id_2	user_id_3	user_id_4	user_id_5	user_id_6	user_id_7	user_id_8	user_id_9	...	item_id_1673	item_id_1674
0	False	0	0	0	0	0	0	0	0	0	...	0	0
1	False	0	0	0	0	0	0	0	0	0	...	0	0
2	False	0	0	0	0	0	0	0	0	0	...	0	0
3	False	0	0	0	0	0	0	0	0	0	...	0	0
4	False	0	0	0	0	0	0	0	0	0	...	0	0

5 rows × 2626 columns

Learning with Sparse Data - Example

```
y = data_processed["rating"]
X = data_processed.drop(["rating"],axis=1)
sparsity = pd.Series(np.count_nonzero(X, axis=0)*100 / x.shape[0],
                     index = X.columns)
sparse_filtered = sparsity.sort_values(ascending=False)[:10]
ax = sparse_filtered.plot.barh(title="Top 10 populated features")
ax.set_xlabel("% populated")
```

```
Text(0.5, 0, '% populated')
```



Learning with Sparse Data - Example

```
dev_X, test_X, dev_y, test_y = train_test_split(X, y, test_size=0.2,
                                                random_state=42)
lr_model = LogisticRegression()
lr_model.fit(dev_X, dev_y)
display(lr_model.score(dev_X, dev_y))
display(lr_model.score(test_X, test_y))
```

```
0.7306125
0.70845
```

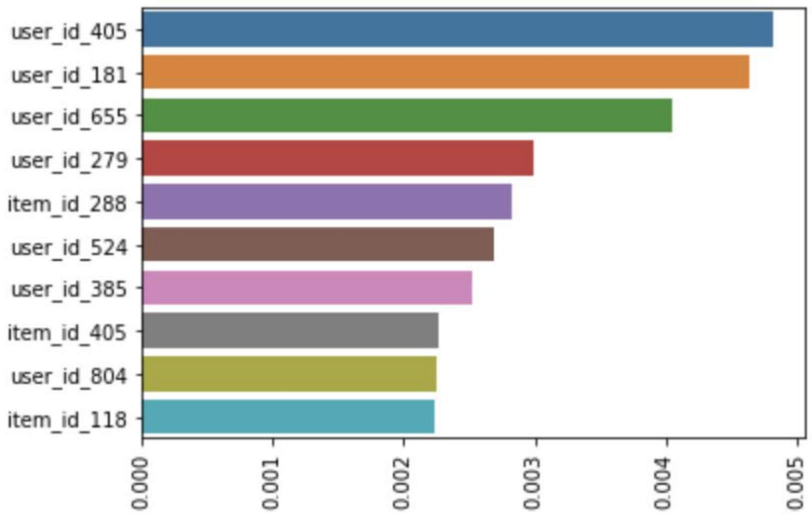
Learning with Sparse Data - Example

```
dev_X, test_X, dev_y, test_y = train_test_split(X, y, test_size=0.2,
                                                random_state=42)
model = DecisionTreeClassifier()
model.fit(dev_X, dev_y)
display(model.score(dev_X, dev_y))
display(model.score(test_X, test_y))
```

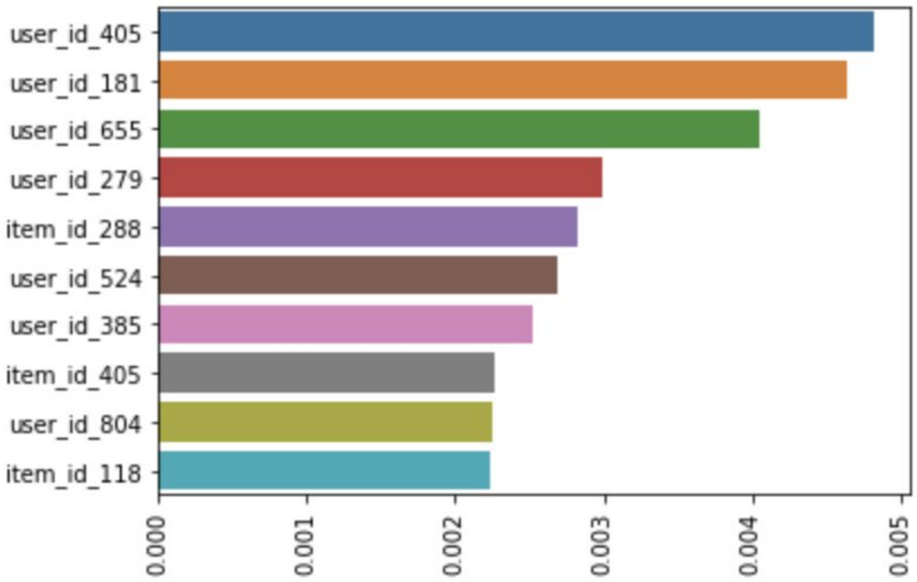
```
1.0
0.68655
```

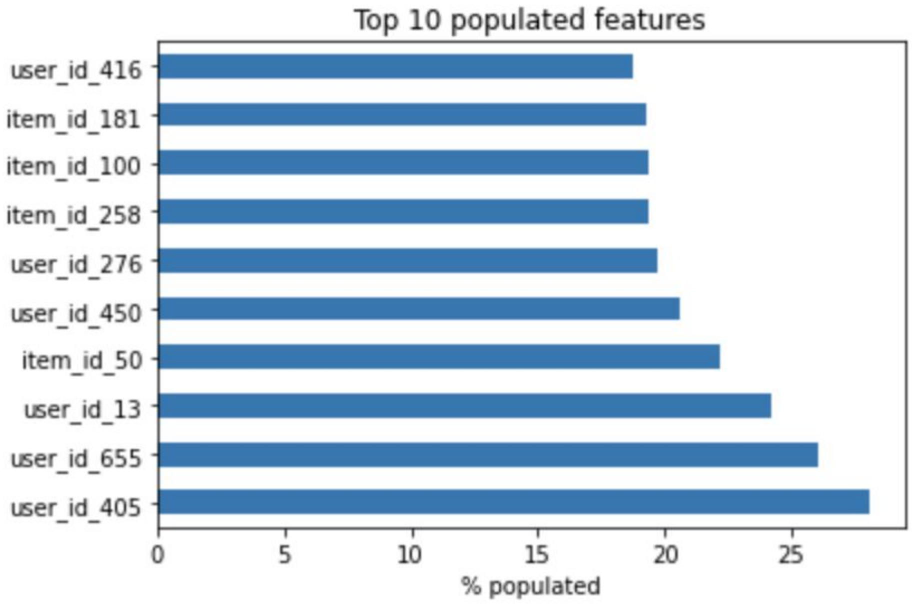
Learning with Sparse Data - Example

```
feat_imps = list(zip(dev_X.columns, model.feature_importances_))
feats, imps = zip(*(sorted(list(filter(lambda x: x[1] != 0, feat_imps)),
                         key=lambda x: x[1], reverse=True)))
ax = sns.barplot(list(imps[:10]), list(feats[:10]))
ax.tick_params(axis='x', rotation=90)
```



Learning with Sparse Data - Example





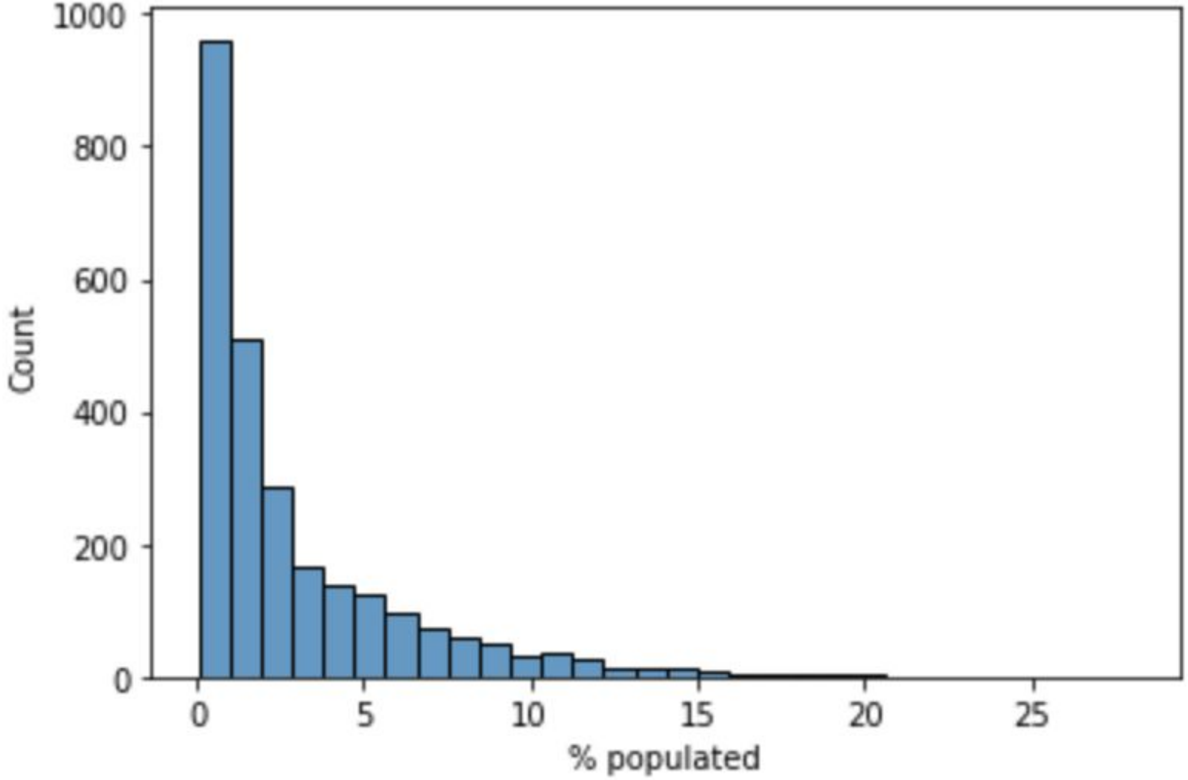
Learning with Sparse Data

- Feature selection
- Feature transformation
- Embedded methods

Learning with Sparse Data - Feature selection

- Sparsity-based feature selection
- Variance-based feature selection
- Univariate feature selection

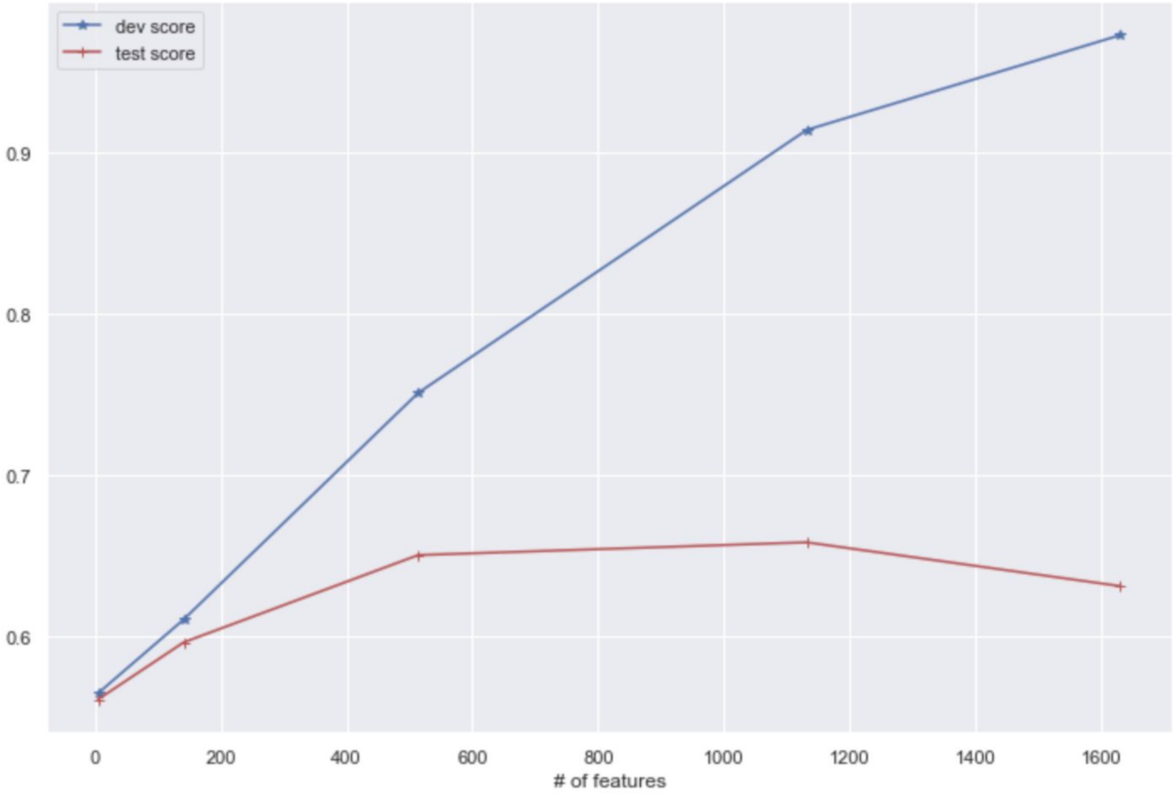
Learning with Sparse Data - Sparsity-Based Filtering



Learning with Sparse Data - Sparsity-Based Filtering

```
sparsities = [1, 2, 5, 10, 20]
dev_scores = []
test_scores = []
n_features = []
for val in sparsities:
    sparsity_filter = sparsity >= val
    dev_X_filtered = dev_X[dev_X.columns[sparsity_filter]]
    test_X_filtered = test_X[test_X.columns[sparsity_filter]]
    model = DecisionTreeClassifier()
    model.fit(dev_X_filtered, dev_y)
    n_features.append(np.sum(sparsity_filter))
    dev_scores.append(model.score(dev_X_filtered, dev_y))
    test_scores.append(model.score(test_X_filtered, test_y))
```

Learning with Sparse Data - Sparsity-Based Filtering



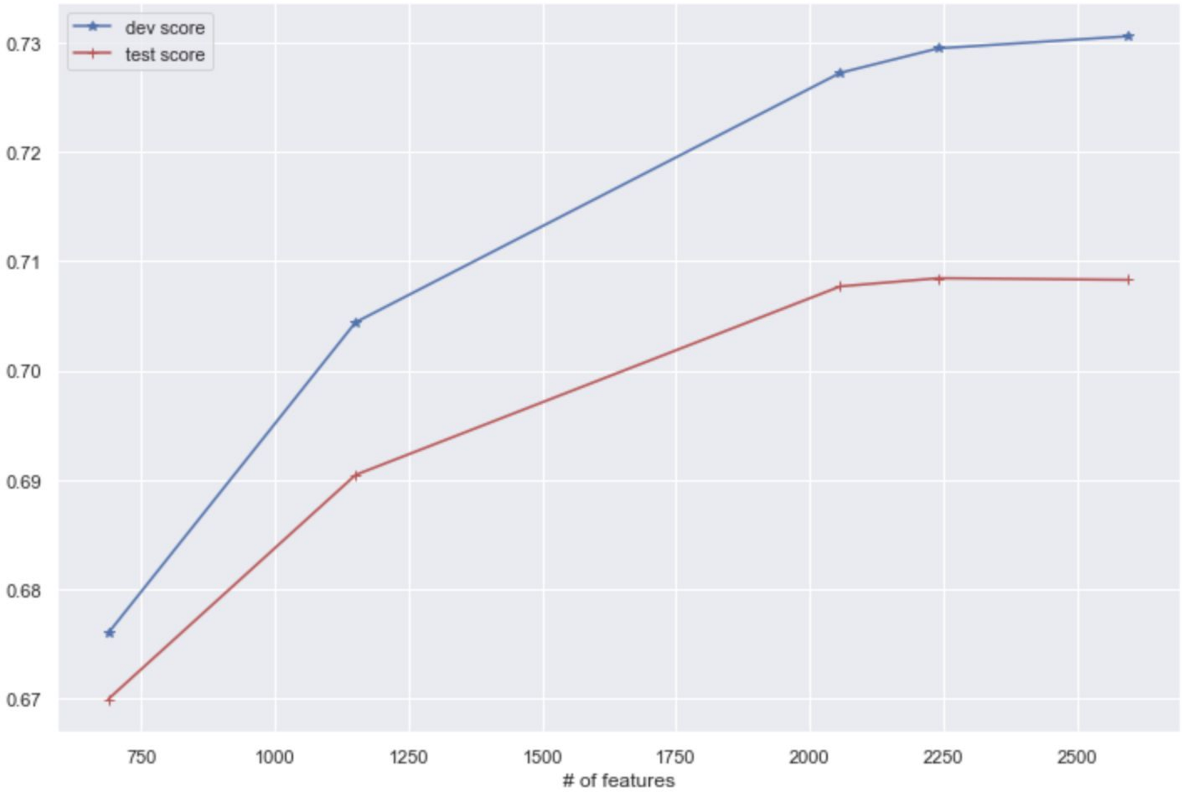
Learning with Sparse Data - Variance-Based Filtering

- Variance-based filtering is a simple baseline filtering technique for feature selection
- It removes features that don’t meet a variance threshold
- Features with most values being the same are generally filtered using this technique.

Learning with Sparse Data - Variance-Based Filtering

```python
var_thresholds = [1e-5, 5e-5, 1e-4, 5e-4, 1e-3]
dev_scores = []
test_scores = []
n_features = []
for thresh in var_thresholds:
    pipe = make_pipeline(VarianceThreshold(thresh), LogisticRegression())
    pipe.fit(dev_X, dev_y)
    n_features.append(len(pipe.named_steps["variancethreshold"].get_support(indices=True)))
    dev_scores.append(pipe.score(dev_X, dev_y))
    test_scores.append(pipe.score(test_X, test_y))
```

Learning with Sparse Data - Variance-Based Filtering



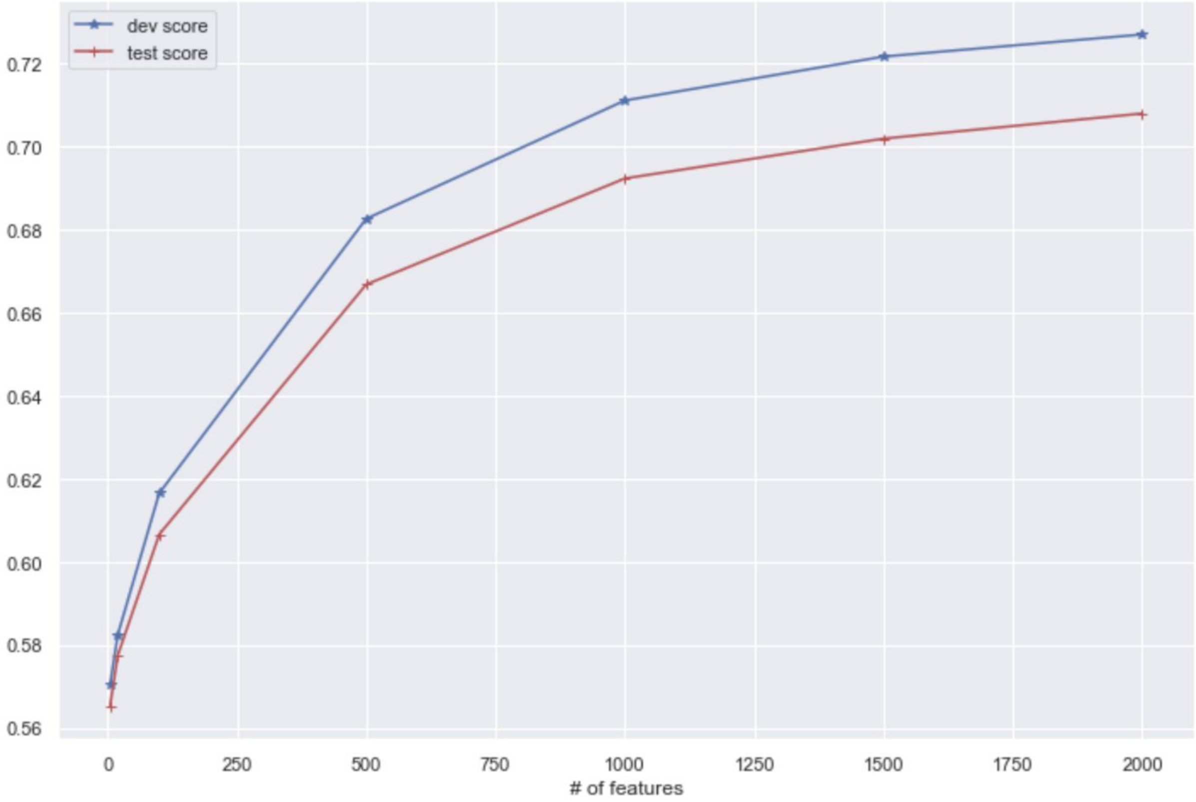
Learning with Sparse Data - Univariate Feature Selection

- Univariate feature selection techniques work by selecting the best features based on univariate statistical tests between feature & target.
- Based on the target (classification or regression), different statistical measures (chi2-statistic, F-statistic etc.) are used.

Learning with Sparse Data - Univariate Feature Selection

```
n_features = [5, 20, 100, 500, 1000, 1500, 2000]
dev_scores = []
test_scores = []
for features in n_features:
    pipe = make_pipeline(SelectKBest(chi2, k=features), LogisticRegression())
    pipe.fit(dev_X, dev_y)
    dev_scores.append(pipe.score(dev_X, dev_y))
    test_scores.append(pipe.score(test_X, test_y))
```

Learning with Sparse Data - Univariate feature selection



Learning with Sparse Data - Feature Transformation

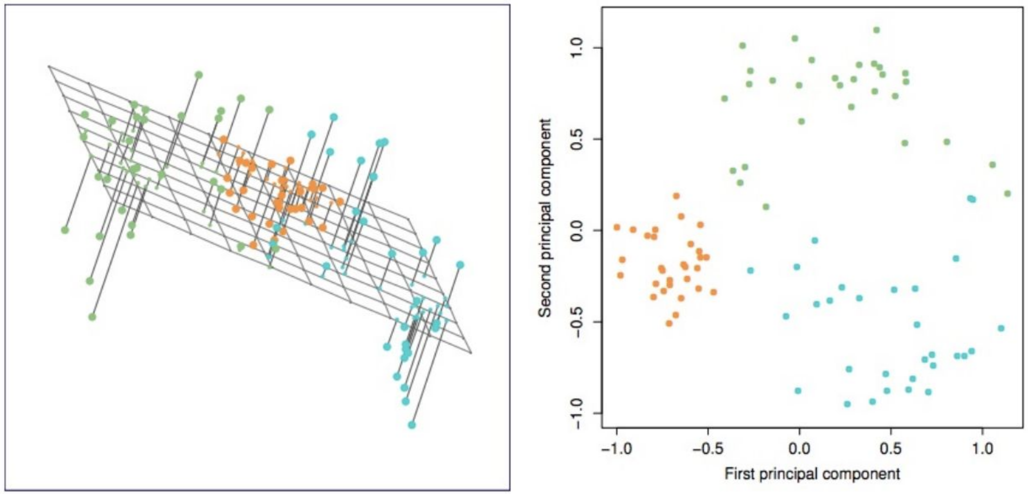
- Feature transformation techniques involve transforming the original feature space to a transformed feature space.
- The transformation generally involves projecting to a low-dimensional space that preserves some of the properties of the data.

Learning with Sparse Data - Feature Transformation

- Principal Component Analysis (PCA)
- Feature hashing

Learning with Sparse Data - Principal Component Analysis

Principal component analysis (PCA) projects data to a lower-dimensional linear subspace, in a way that preserves the axes of highest variance in the data.

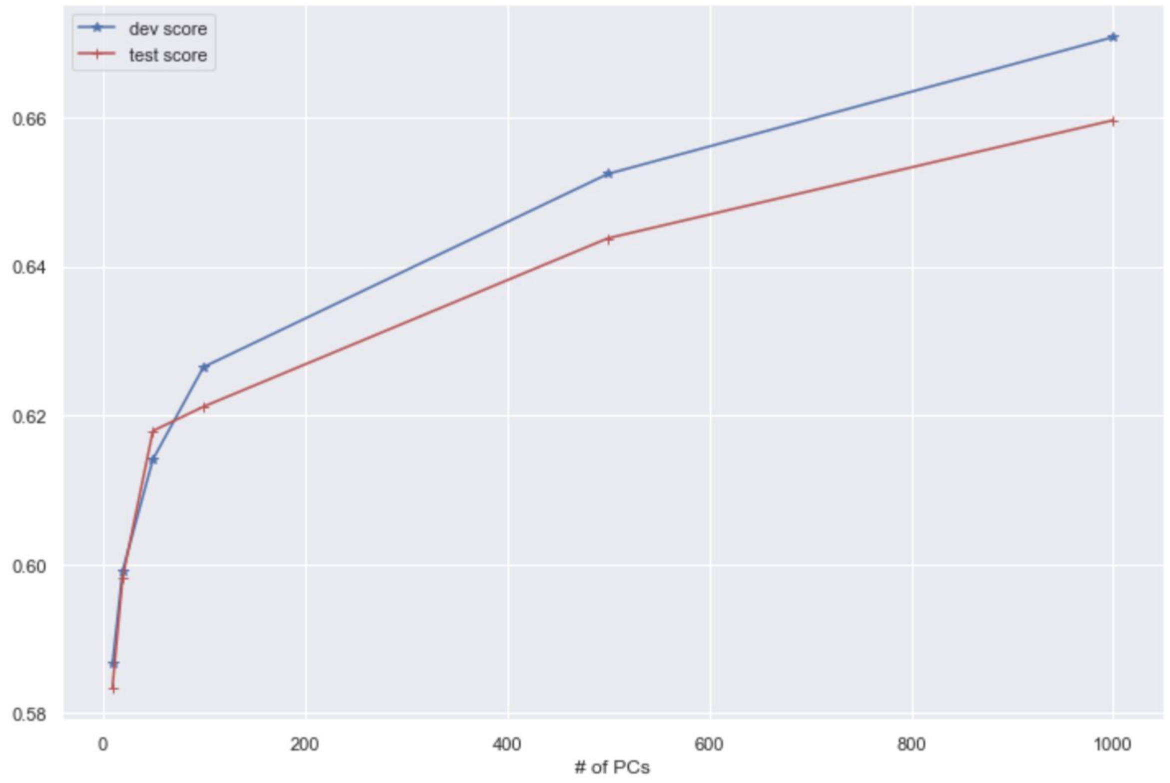


Source: An Introduction to Statistical Learning, Witten et al.

Learning with Sparse Data - Principal Component Analysis

```
scaler = StandardScaler()
dev_X_transformed = scaler.fit_transform(dev_X)
test_X_transformed = scaler.transform(test_X)
n_pc = [10, 20, 50, 100, 500, 1000]
dev_scores = []
test_scores = []
for pc in n_pc:
    pipe = make_pipeline(PCA(n_components=pc), LogisticRegression())
    pipe.fit(dev_X_transformed, dev_y)
    dev_scores.append(pipe.score(dev_X_transformed, dev_y))
    test_scores.append(pipe.score(test_X_transformed, test_y))
```

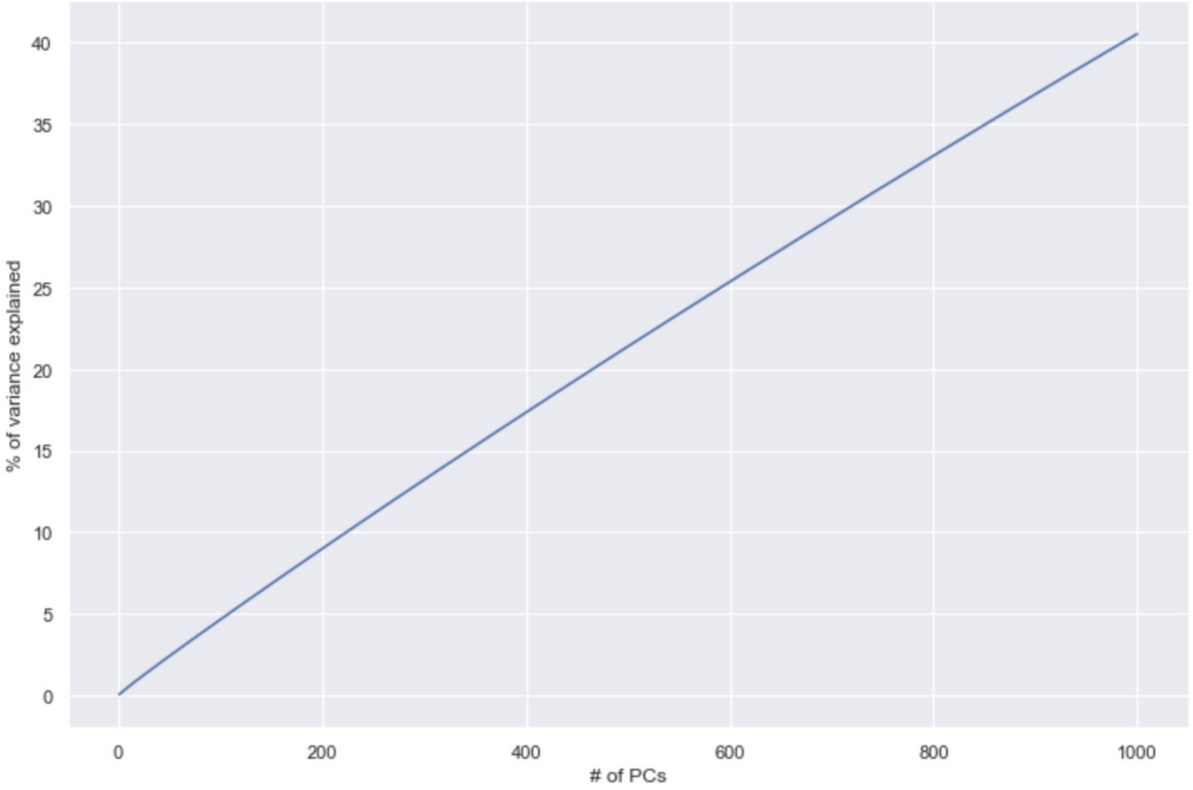
Learning with Sparse Data - Principal Component Analysis



Learning with Sparse Data - Principal Component Analysis

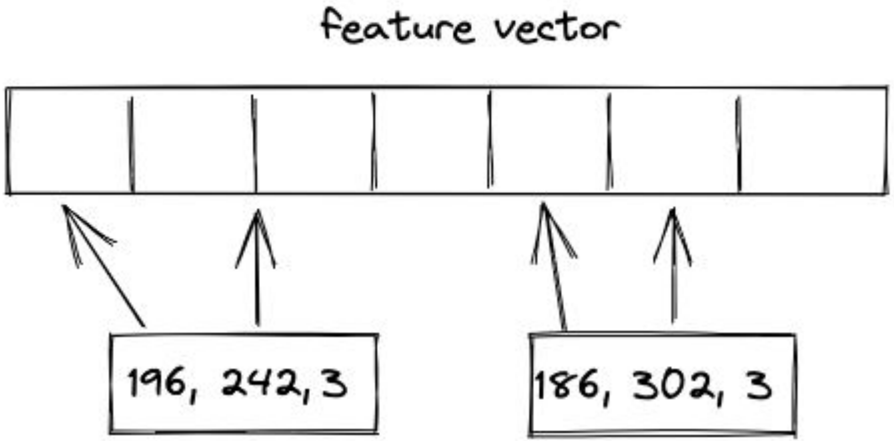
```
pca = PCA(n_components=1000)
dev_X_pc = pca.fit_transform(dev_X_transformed)
sns.set()
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(1, 1, 1)
plt.plot(np.arange(1, 1001), pca.explained_variance_ratio_.cumsum()*100)
ax.set_xlabel("# of PCs")
ax.set_ylabel("% of variance explained")
```

Learning with Sparse Data - Principal Component Analysis



Learning with Sparse Data - Feature Hashing

- Feature hashing is a fast and space-efficient way to vectorize features i.e. converting features to indices in a vector/array
- The conversion is typically achieved using hash functions (murmur3, MD5, SHA1, SHA2, etc.)



Learning with Sparse Data - Feature Hashing

```
data = pd.read_csv('ml-100k/u.data', sep="\t", header=None)
data.columns = ["user_id", "item_id", "rating", "timestamp"]
data.drop(["timestamp"], axis=1, inplace=True)
display(data.head())
display(data.shape)
```

	user_id	item_id	rating
0	196	242	3
1	186	302	3
2	22	377	1
3	244	51	2
4	166	346	1

```
(100000, 3)
```

```
X = data.drop(["rating"],axis=1)
X_prep = list(map(lambda x: dict(zip([str(y) for y in x], [1, 1])),
                  X.values.tolist()))
X_prep[:5]
```

```
[{'196': 1, '242': 1},
 {'186': 1, '302': 1},
 {'22': 1, '377': 1},
 {'244': 1, '51': 1},
 {'166': 1, '346': 1}]
```

Learning with Sparse Data - Feature Hashing

```
fh = FeatureHasher(n_features=10, alternate_sign=False)
X_hashed = fh.transform(X_prep)
y = data["rating"]
X_hashed.todense()
```

```
matrix([[1., 0., 0., ..., 0., 0., 0.],
        [1., 0., 0., ..., 0., 0., 1.],
        [0., 1., 0., ..., 0., 1., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 1.],
        [1., 0., 0., ..., 1., 0., 0.]])
```

Learning with Sparse Data - Feature Hashing

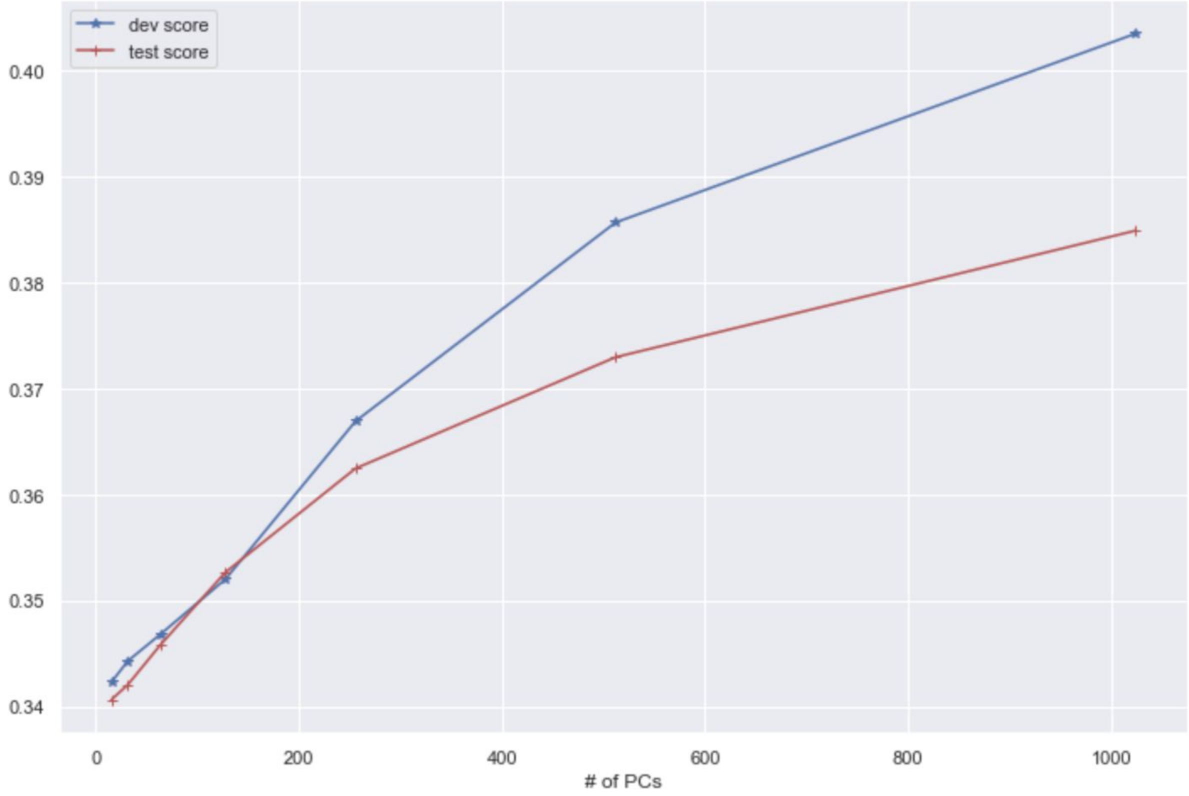
```
dev_X, test_X, dev_y, test_y = train_test_split(X_hashed, y, test_size=0.2,
                                                    random_state=42)
lr_model = LogisticRegression()
lr_model.fit(dev_X, dev_y)
display(lr_model.score(dev_X, dev_y))
display(lr_model.score(test_X, test_y))
```

```
0.4034875
0.3849
```

Learning with Sparse Data - Feature Hashing

```
n_features = [16, 32, 64, 128, 256, 512, 1024]
dev_scores = []
test_scores = []
for features in n_features:
    fh = FeatureHasher(n_features=features, alternate_sign=False)
    X_hashed = fh.transform(X_prep)
    dev_X, test_X, dev_y, test_y = train_test_split(X_hashed, y, test_size=0.2,
                                                    random_state=42)
    pipe = make_pipeline(SelectKBest(chi2, k=features), LogisticRegression())
    pipe.fit(dev_X, dev_y)
    dev_scores.append(pipe.score(dev_X, dev_y))
    test_scores.append(pipe.score(test_X, test_y))
```

Learning with Sparse Data - Feature Hashing



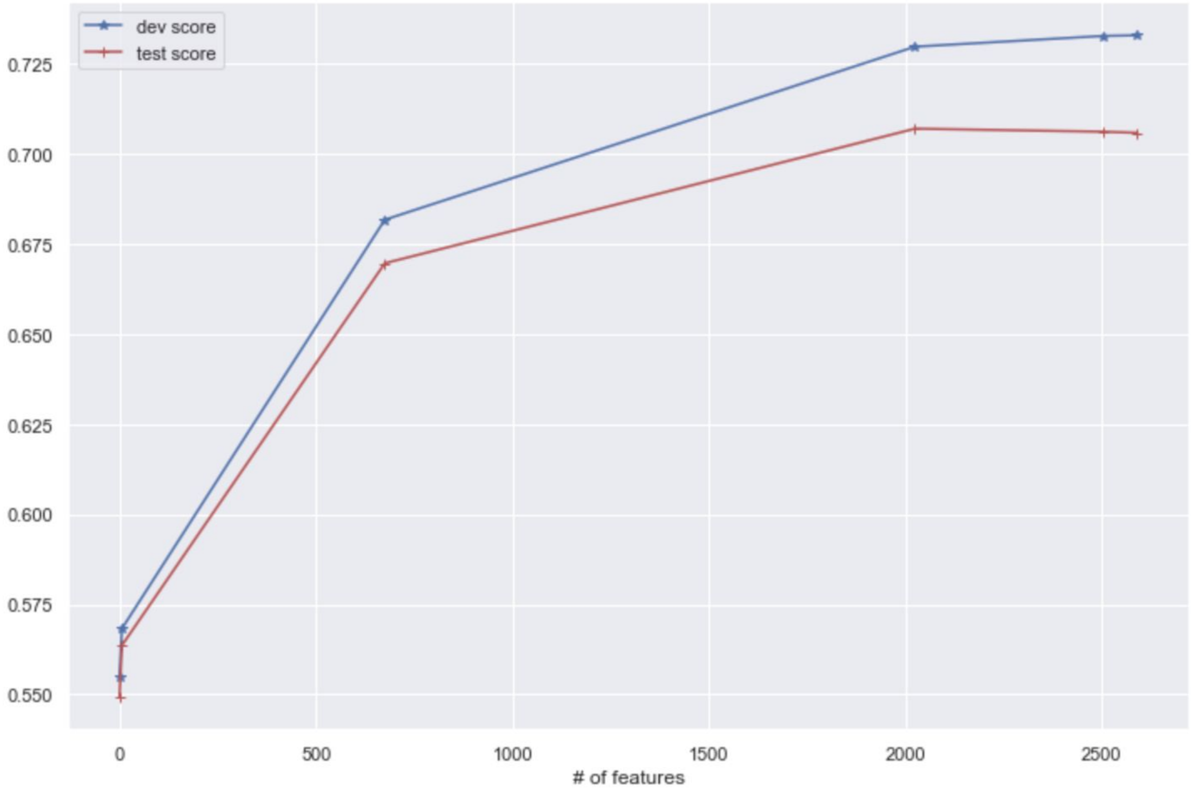
Learning with Sparse Data - Embedded methods

- So far, we have tackled sparse data by reducing the feature space (either through feature selection and/or feature extraction) and then training a classifier
- Some methods allow you to combine these steps (i.e. dimensionality reduction and model training)
- Some examples for embedded methods include L1-variants of classifiers as well as elastic-net variants.

Learning with Sparse Data - Lasso Logistic Regression

```
C_values = [0.001, 0.01, 0.1, 1, 10, 100]
dev_scores = []
test_scores = []
n_features = []
for C in C_values:
    lr_model = LogisticRegression(C=C,
                                  penalty='l1',
                                  solver='liblinear')
    lr_model.fit(dev_X, dev_y)
    n_features.append(np.count_nonzero(lr_model.coef_))
    dev_scores.append(lr_model.score(dev_X, dev_y))
    test_scores.append(lr_model.score(test_X, test_y))
```

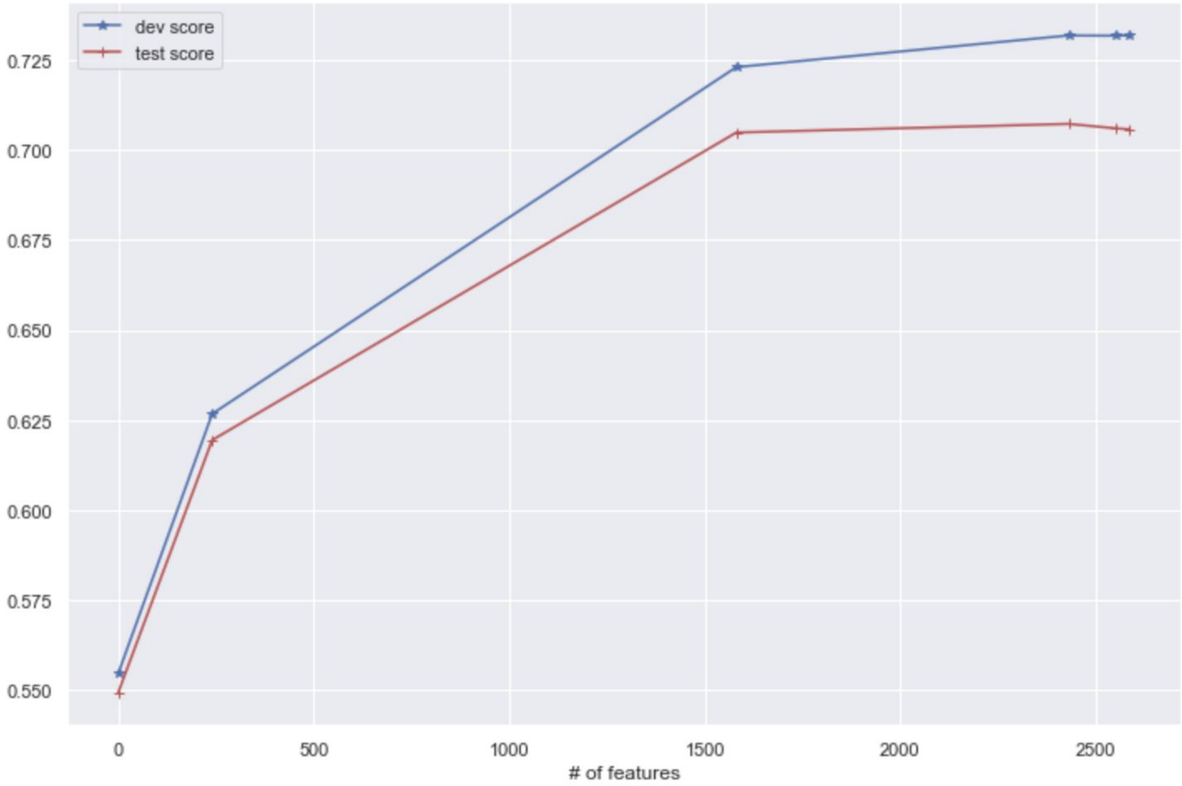
Learning with Sparse Data - Lasso Logistic Regression



Learning with Sparse Data - Lasso SVMs

```
C_values = [0.001, 0.01, 0.1, 1, 10, 100]
dev_scores = []
test_scores = []
n_features = []
for C in C_values:
    svc_model = LinearSVC(C=C, penalty='l1', dual=False)
    svc_model.fit(dev_X, dev_y)
    n_features.append(np.count_nonzero(svc_model.coef_))
    dev_scores.append(svc_model.score(dev_X, dev_y))
    test_scores.append(svc_model.score(test_X, test_y))
```

Learning with Sparse Data - Lasso SVMs



Learning with Sparse Data - Methods not covered

- Feature selection
  - Other univariate selection techniques
- Feature transformation
  - Kernel PCAs
  - Non-linear dimensionality reduction (LLEs, Isomaps etc.)
- Embedded methods
  - Tree-based methods (Decision Trees, Random Forests, Gradient Boosted Trees)
- Wrapper methods
  - Support Vector Machines-Recursive Feature Elimination (SVM-RFEs)

Questions?