Predicting the Weather: Is it Windy?

It is quite hard. Step Outside

Objective

Determine if we can accurately predict the wind speed in St. Louis based on a number of parameters.

Assumption that weather is not location based in a given moment.

- For example, the weather is dictated by climate and latitude, but a given moment can be really anything. Only the likelihood of certain conditions (ie. snow) would be greater or smaller. This does not matter in our modeling.

Data Import

Training data came from the Iowa State University weather station.

Forecasting data provided by weather.gov's api.

```python
# Replace with your desired latitude and longitude
latitude = '38.685066'
longitude = '-90.349317'

# Construct API endpoint URL
url = f'https://api.weather.gov/points/{latitude},{longitude}'

# Construct the API endpoint URL for the given point
point_url = f'https://api.weather.gov/points/{latitude},{longitude}'

# Make a GET request to the API for the given point
point_response = requests.get(point_url)

# Check if the request was successful
if point_response.status_code == 200:

    # Parse the JSON response
    point_data = point_response.json()

    # Extract the hourly forecast URL
    hourly_forecast_url = point_data['properties']['forecastHourly']

    # Make a GET request to the hourly forecast URL
    hourly_forecast_response = requests.get(hourly_forecast_url)

    # Check if the hourly forecast request was successful
    if hourly_forecast_response.status_code == 200:
        # Parse the JSON response
```

	station	valid	tmpf	dwpf	relh	drct	sknt	p01i	alti	mslp	...	wxcodes	ice_accretion_1hr	ice_accretion_3hr	ice_accretion_6hr	peak_wind_gust	peak_wind_drct	peak_wind_time	feel	metar	snowdepth
0	IKV	2023-01-01 00:15	35.60	33.80	93.08	0.00	0.00	0.0	29.75	M	...	M	M	M	M	M	M	M	35.60	KIKV 010015Z AUTO 00000KT 10SM CLR 02/01 A2975...	M
1	IKV	2023-01-01 00:35	35.60	32.00	86.59	120.00	4.00	0.0	29.75	M	...	M	M	M	M	M	M	M	31.62	KIKV 010035Z AUTO 12004KT 10SM BKN100 BKN120 0...	M
2	IKV	2023-01-01 00:55	33.80	32.00	93.03	130.00	5.00	0.0	29.75	M	...	M	M	M	M	M	M	M	28.53	KIKV 010055Z AUTO 13005KT 9SM OVC100 01/00 A29...	M
3	IKV	2023-01-01 01:15	33.80	33.80	100.00	0.00	0.00	0.0	29.76	M	...	M	M	M	M	M	M	M	33.80	KIKV 010115Z AUTO 00000KT 9SM FEW075 OVC100 01...	M
4	IKV	2023-01-01 01:35	33.80	32.00	93.03	40.00	4.00	0.0	29.76	M	...	M	M	M	M	M	M	M	29.52	KIKV 010135Z AUTO 04004KT 9SM FEW075 OVC100 01...	M

5 rows × 30 columns

Designing Communicable data

Due to having two data sources, the data from each source needed to be adjusted to match formats.

- For example, the wind speed in one data set was “9 mph” and “9” in another. This was corrected.

We then had to combine the tables so we can train_test_split.

```python
# Map out the Short Forecast to match the other column
def mapping_forecast(forecast):
    mapping = {'Mostly Clear': 'FEW',
        'Partly Cloudy': 'SCT',
        'Clear': 'CLR',
        'Sunny': 'CLR',
        'Mostly Sunny': 'FEW',
        'Partly Sunny': 'SCT',
        'Mostly Cloudy': 'BKN',
        'Cloudy': 'OVC',
        'Chance Rain Showers': 'SCT',
        'Slight Chance Light Rain': 'SCT',
        'Chance Light Rain': 'SCT',
        'Light Rain Likely': 'BKN'
    }
    return mapping.get(forecast, 'Unknown')

forecast_weather['Short Forecast'] = forecast_weather['Short Forecast'].apply(mapping_forecast)

# Drop 'Unknown' and 'W'
forecast_weather = forecast_weather[~forecast_weather['Short Forecast'].isin(['Unknown', 'W'])]
forecast_weather.head()
```

✓ 0.0s

```python
forecast  = []

periods_data = hourly_forecast_data['properties']['periods']
for period in periods_data:
    forecast.append({
        'Temperature':period['temperature'],
        'Dewpoint': period['dewpoint']['value'],
        'Relative Humidity': period['relativeHumidity']['value'],
        'Probability of Percipitation':period['probabilityOfPrecipitation']['value'],
        'Short Forecast': period['shortForecast'],
        'Wind Speed': period['windSpeed'],
#         'Is Daytime':period['isDaytime'],
    })
forecast_weather = pd.DataFrame(forecast)
forecast_weather.head()
```

✓ 0.0s

Normalizing the data

Standard Scaling and One Hot Encoding...

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
complete_df[['Temperature','Dewpoint','Relative Humidity','Probability of Percipitation']] = scaler.fit_transform(complete_df[['Temperature','Dewpoint','Relative Humidity','Probability of Percipitation']])
complete_df
```

✓ 0.0s

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(drop='first')
complete_df =pd.get_dummies(complete_df, columns = ['Short Forecast'], drop_first = True, dtype=float)
complete_df
```

✓ 0.0s

We did it.

Train, Test, Split

We had to split a little odd because we had to manually train test split our data because our historic data and forecast data needed to be the split between historic and forecast, not randomly as train_test_split would usually do.

```
X_train = complete_df[complete_df['H/F'] == 'H'].drop(columns = ['Wind Speed (Knots)','H/F'])
X_test = complete_df[complete_df['H/F'] == 'F'].drop(columns = ['Wind Speed (Knots)','H/F'])
y_train = complete_df[complete_df['H/F'] == 'H']['Wind Speed (Knots)']
y_test = complete_df[complete_df['H/F'] == 'F']['Wind Speed (Knots)']
```

Machine Learning Algorithms

We chose to use an XGBoost Regression algorithm as it allows for non-linear regressions and, given our historic data size of over 20,000 rows, a boosted regression generally works faster.

Grid searching was used as hyperparameter tuning to identify optimal model.

```python
from xgboost import XGBRegressor
```

✓ 0.0s

```python
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of boosting rounds
    'learning_rate': [0.01, 0.1, 0.2],  # Step size shrinkage
    'max_depth': [3, 4, 5],  # Maximum depth of the trees
    'min_child_weight': [1, 2, 3],  # Minimum sum of instance weight needed in a child
    'subsample': [0.8, 0.9, 1.0],  # Fraction of samples used for training
    'colsample_bytree': [0.8, 0.9, 1.0],  # Fraction of features used for training each tree
}

#Initialize GridSearchCV with the model and parameter grid
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='neg_mean_squared_error', cv=5, n_jobs=-1)

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Fit the best model to the data
best_model.fit(X_train, y_train)
```

```
Best Hyperparameters: {'colsample_bytree': 1.0, 'learning_rate': 0.01, 'max_depth': 3, 'min_child_weight': 1, 'n_estimators': 300, 'subsample': 0.8}
```

```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=1.0, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.01, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=300, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

```python
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Predict on the test data
y_preds = best_model.predict(X_test)
round_preds = np.round(y_preds)

# Calculate regression metrics
mse = mean_squared_error(y_test, round_preds)
mae = mean_absolute_error(y_test, round_preds)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, round_preds)

print("Mean Squared Error (MSE):", mse)
print("Mean Absolute Error (MAE):", mae)
print("Root Mean Squared Error (RMSE):", rmse)
print("R-squared (R²):", r2)
```

```
Mean Squared Error (MSE): 7.090277777777778
Mean Absolute Error (MAE): 2.1041666666666665
Root Mean Squared Error (RMSE): 2.6627575514450763
R-squared (R²): -0.45079928952042625
```

Results

A negative R2 value indicates that the model cannot explain any variation based on the training data.

Neural Network?

Started with a very basic Neural Network. Making small changes to see what works and what doesn't work.

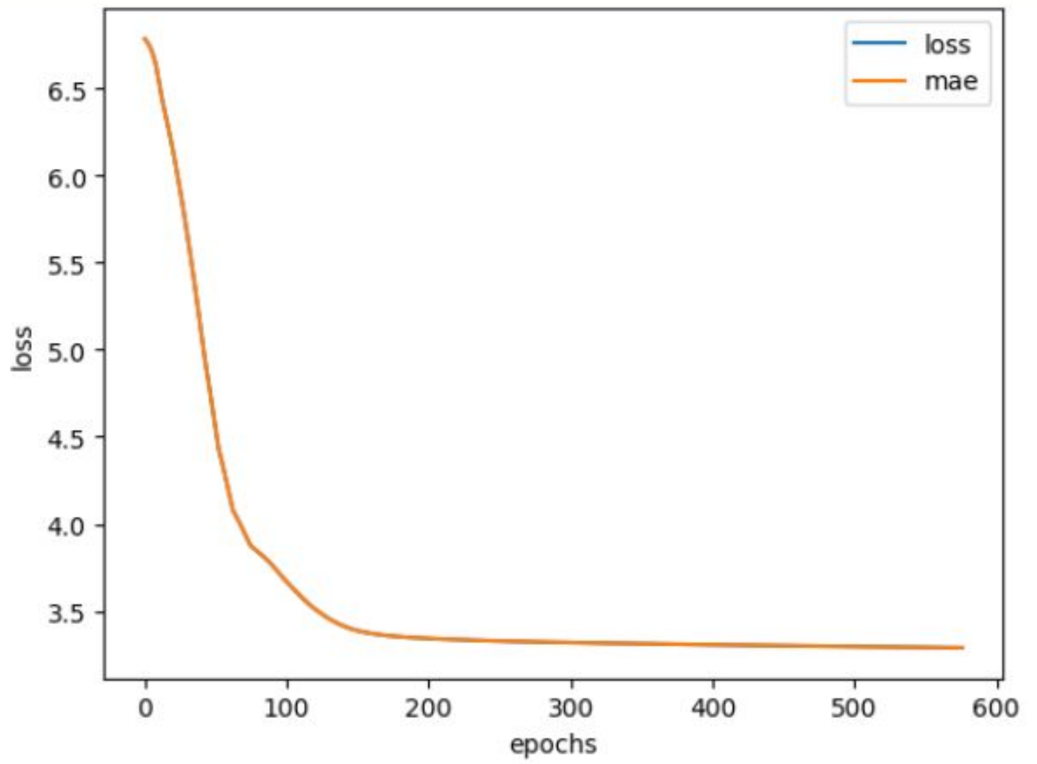
```
#set the random seed
tf.random.set_seed(42)

#set a callback to earlystop model
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

#build a model
weather_model = tf.keras.Sequential([
    tf.keras.layers.Dense(2),
    tf.keras.layers.Dense(1),
    tf.keras.layers.Dense(1)
])

#compile the model
weather_model.compile(loss=tf.keras.losses.mae,
                      optimizer=tf.keras.optimizers.Adam(),
                      metrics=['mae'])

#fit the model to a variable history for plotting
history = weather_model.fit(X_train_normal, y_train, epochs=600, callbacks=callback, verbose=1)
```

✓ 17.8s

Neural Network 2

Made a total of 16 experiments, involving adding/removing

- Hidden layers
- Neurons
- Activation:
- Kernel_regularizer
- Loss function:
- Optimizer functions
- Epochs: 1000
- Callbacks: Earlystopping



The end of the Neural Network... Keras Tuner

Utilizing KerasTuner

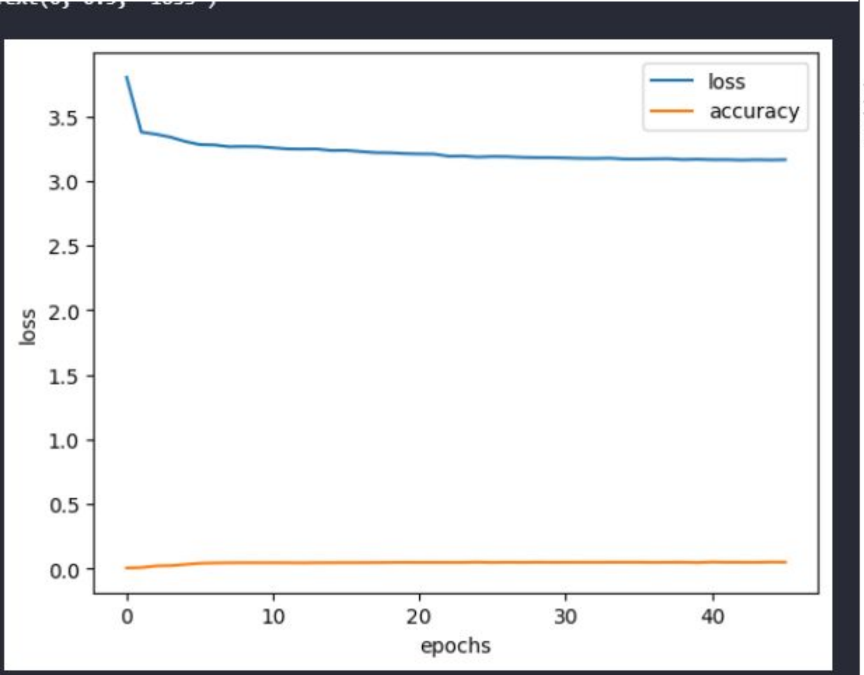
```
import keras_tuner as kt

#building keras tuner
def model(hp):

    model = tf.keras.Sequential()

    #set the activation functions
    activation = hp.Choice('activation', ['relu','tanh','swish'])

    hp_learning_rate = hp.Choice('learning_rate', values=[1e-1, 1e-2, 1e-3, 1e-4])

    model.add(tf.keras.layers.Dense(units=hp.Int('units',
                                                  min_value=2,
                                                  max_value=512,
                                                  step=16),
                                                  activation=activation,
                                                  input_dim=9
                                                  ))

    #Allow kerastuner to decide number of hidden layers and neurons in hidden
    for i in range(hp.Int('num_layers', 1, 5)):
        model.add(tf.keras.layers.Dense(units=hp.Int('units_' + str(i),
                                                      min_value=1,
                                                      max_value=30,
                                                      step=1),
                                                      activation=activation,
                                                      kernel_regularizer=tf.keras.regularizers.l1(l=hp.Choice('l1_weight', [0.1, 0.01, 0.001, 0.0001]))
                                                      ))
    #this is my drop out layer
    model.add(tf.keras.layers.Dropout(hp.Choice('dropout', [0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
                                          ))

    #final layer output
    model.add(tf.keras.layers.Dense(1))

    #compile the model
    model.compile(loss=tf.keras.losses.mae,
                  optimizer=tf.keras.optimizers.RMSprop(learning_rate=hp_learning_rate),
                  metrics=['accuracy'])

    return model
```

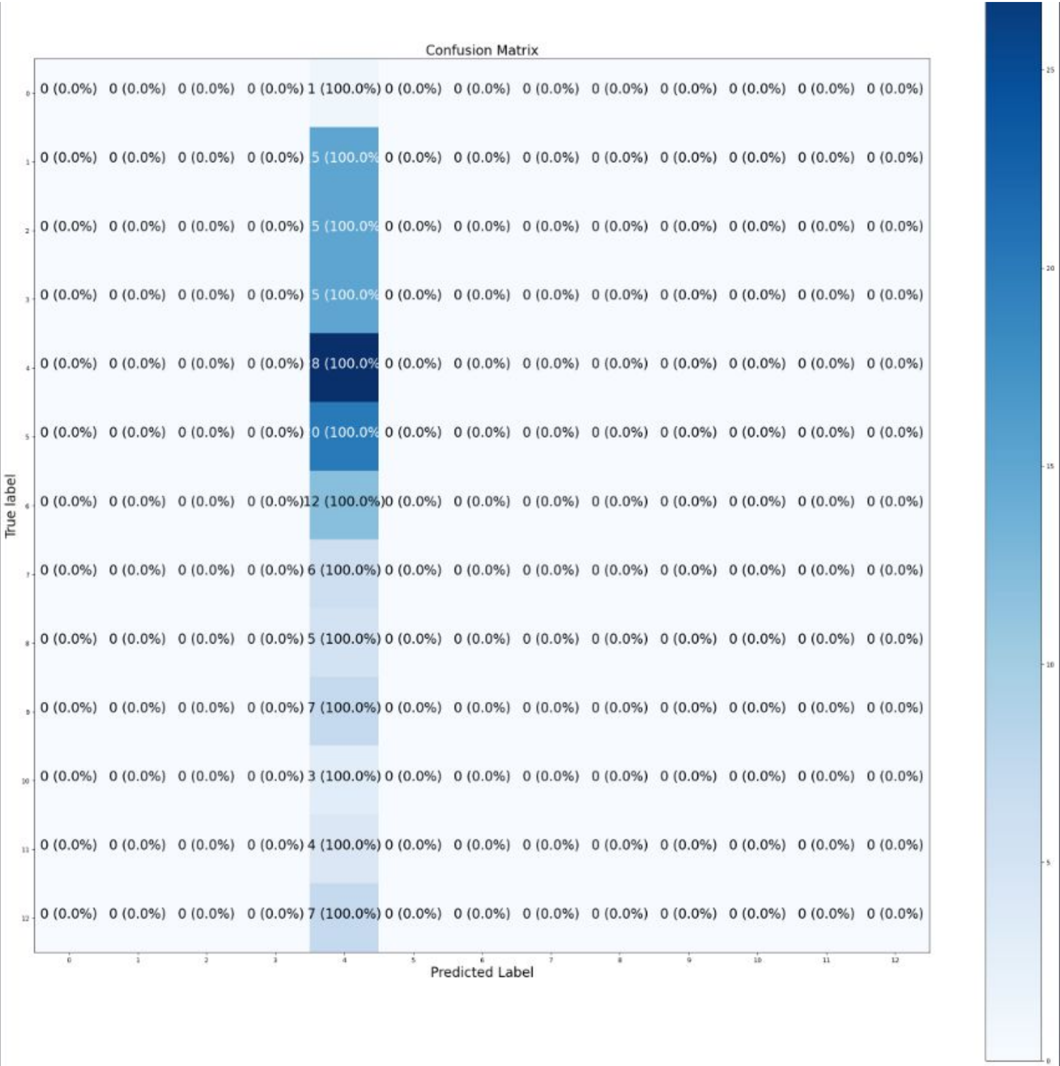
Results 2.0

Unable to achieve model learning.

Need to try other (i.e. models Convoluted Network.)

Better features





Final Conclusions

Using a normal regression function instead of a boosted regression function may allow for better accuracy at the expense of significantly more time training the model.

Adding more columns of data might improve accuracy (time of day, season, different cloud coverage at different levels, etc.)

Likely using a deep learning models on cloud coverage would be optimal.

Change to classification model for predicting winds speeds within buckets (0-10mph, 11-20mph, etc.) since small differences in wind speed do not matter. This likely would result in better accuracy but less precision.